MASTER IN COMPUTER SCIENCE

A multi-objective genetic algorithm for biclustering of gene expression data with probabilistic encoding and overlapping control

Marcozzi, Michaël

Award date:
2010

Awarding institution:
University of Namur

MASTER OF COMPUTER SCIENCE

A multi-objective genetic algorithm for biclustering of gene expression data with probabilistic encoding and overlapping control

Michaël Marcozzi

2012

University of Namur

Facultés Universitaires Notre-Dame de la Paix, Namur
Faculté d'informatique
Année académique 2009-2010

A Multi-Objective genetic algorithm for Biclustering of gene expression data with Probabilistic Encoding and Overlapping Control

Michaël MARCOZZI

Mémoire présenté en vue de l'obtention du grade de
Master en Sciences Informatiques

Résumé. Le travail de recherche présenté dans ce mémoire se situe au carrefour de l'informatique et de la biologie. Il consiste dans le développement d'un programme informatique capable d'extraire des informations utiles hors de données biologiques, en utilisant une stratégie algorithmique inspirée par le modèle biologique de l'évolution des espèces.
Le bipartitionnement de données d'expression de gènes consiste à analyser de grandes quantités de données d'expériences de biologie mesurant le niveau d'expression de certains gènes dans certaines conditions expérimentales (différents tissus, différents patients...), afin d'identifier des groupes de gènes qui présentent des comportements cohérents sous certains groupes de conditions. De telles corrélations peuvent être un indice de l'existence d'une relation biologique entre les gènes et les conditions identifiées, ce qui peut se révéler une information particulièrement intéressante d'un point de vue biologique et médical.
Les algorithmes évolutionnaires constituent une classe générique d'algorithmes, qui ont en commun l'utilisation de mécanismes inspirés par l'évolution darwinienne, afin de résoudre des problèmes dont les meilleures solutions doivent être découvertes selon certains critères de qualité fixés. Les techniques d'algorithmique évolutionnaire peuvent être particulièrement bien adaptées pour réaliser un bipartitionnement efficace de données d'expression de gènes, et plusieurs approches évolutionnaires de bipartitionnement ont été proposées dans la littérature.
Dans ce mémoire, nous présentons MOBPEOC, une nouvelle approche évolutionnaire de bipartitionnement que nous avons développée, afin d'améliorer le mécanisme de bipartitionnement d'une approche évolutionnaire existante. En particulier, MOBPEOC représente un premier test grandeur nature pour une nouvelle technique évolutionnaire à portée générale, appelée encodage probabiliste, et que nous proposons pour la première fois dans le cadre de ce travail.
Une évaluation expérimentale de l'algorithme MOPBEOC est proposée, où la technique est mise à l'épreuve sur de véritables données biologiques. La comparaison des résultats obtenus, par rapport à l'approche évolutionnaire de bipartitionnement précédente, montre une forte amélioration de la qualité des solutions découvertes.

Mots-clés. Bio-informatique, Exploration de Données, Bipartitionnement, Données d'Expression de Gènes, Algorithmes Evolutionnaires, Optimisation Multimodale et Multi-Objectifs, Encodage Probabiliste.

Abstract. The research work presented in this thesis stands at the crossroads of computer sciences and biology, as it consists in the development of a computer program to extract useful information from biological data, using an algorithmic strategy inspired by the biological model of the evolution of the species.
Biclustering of gene expression data means analyzing large amounts of biological experimental data measuring the level of expression of some genes under some experimental conditions (different tissues, different patients,...), in order to individuate groups of genes that exhibit coherent behaviors under some groups of conditions. Such correlations may provide a hint over an existing biological relation between the individuated genes and conditions, and thus be of particular interest from a biological and medical point of view.
Evolutionary computation is a generic class of algorithms, sharing the use of mechanisms inspired by darwinian evolution to solve problems whose best solutions, according to some fixed quality criteria, have to be discovered. Evolutionary computation techniques can be particularly relevant to achieve an efficient biclustering of gene expression data, and several evolutionary biclustering approaches have been proposed in the literature.
In this thesis, we present MOBPEOC, a new evolutionary biclustering approach that we developed to improve the biclustering mechanism of an existing evolutionary approach. In particular, MOBPEOC represents a first life-size test for a new general-purpose evolutionary technique that we propose for the first time in this work, called probabilistic encoding.
An experimental evaluation of the MOBPEOC algorithm is proposed, where the technique is applied to real biological data. The comparison of the obtained results with the previous biclustering evolutionary approach shows a strong improvement of the quality of the discovered solutions.

Keywords. Bioinformatics, Data Mining, Biclustering, Gene Expression Data, Evolutionary Computation, Multimodal and Multi-Objective Optimization, Probabilistic Encoding.

Foreword

Building and writing a master thesis is a fight. It has moments of difficulty and despair, and moments of joy and hope. Like any fight, it cannot be won alone. I take thus advantage of these lines to thank all the people who helped me in my quest.

Professor W. Vanhoof has always been present, from the first to the last page of this story. As my promoter, I want to thank him for his availability, his advices and his always fruitful remarks.
Professor F. Divina defined the frame and gave the direction of my work. As my internship supervisor, I want to thank him for the good ideas and advices he proposed and for answering my questions.
S. Amant has been my fellow fighter during many battles. As my internship fellow and friend, I want to thank him for useful discussions.

I thank professor J.-P. Leclercq for the access to several useful book references which particularly helped me in my work.

I also want to thank all those who have been there for me during these months of fight. I particularly thank my family, for giving me the chance to pursue my dreams. I also thank my classmate and friend A. Martin, for her unfailing support.

Contents

Part I

Introduction

Introduction

Computers are machines manipulating symbols. Receiving a set of symbols as input, and another set of symbols as program, they return, after a sequence of purely textual rewriting operations on the input symbols, as described by the program symbols, the set of resulting symbols as output. Most of the magic of computing holds in the fact that these input and output symbols can represent informations about the world. The processing speed of computers and the ubiquity of computer networks allow these informations to be analyzed and transformed quickly in huge proportions, and shared across the world at the speed of light.
Computer science is thus a purely abstract and formal knowledge domain that essentially makes sense when its mechanisms are applied to informations about the world. Two very interesting applications of computer science techniques are data mining and simulation. Data mining takes advantage of the incredible processing speed of computers to extract useful informations from very large amounts of data about the world. Computer simulation consists in writing computer programs that can simulate, in life-size conditions, formal models of processes existing in the real world.

In this thesis, we combine these two applications. The object of the work presented here is the development of a computer program that solves a data mining problem concerned with biological data. This program uses an algorithmic strategy that simulates a formal model of the natural process of evolution of the species.
The data mining problem treated in this work is the gene expression data biclustering problem. In this biological problem, large amounts of experimental data measuring the level of expression of some genes under some experimental conditions should be analyzed, in order to individuate biclusters, i.e. groups of genes that exhibit coherent behaviors under some groups of conditions. Such biclusters may indeed provide a hint over an existing biological relation between the individuated genes and conditions. This information could participate to a better understanding of biological systems at a molecular level, and is thus of particular interest from a biological (like for gene profiling) and medical (like for a better understanding of diseases) point of view.
The biclustering of gene expression data problem can be specified as an optimization problem. Optimization covers the class of problems where the best solutions, according to some fixed quality criteria, have to be found among a set of potential solutions. The gene expression data biclustering problem is a very hard optimization problem, even for a computer. Several biclusters have indeed to be found simultaneously in huge amounts of data, optimizing several antagonist mathematical criteria enforcing the probability for them to have a biological significance.
For solving such hard optimization problems, computer scientists have notably developed generic solutions, inspired by the efficiency of the processes existing in the nature. Evolutionary computation is one of these, grouping a large class of algorithmic optimization

techniques, inspired by computer simulations of the biological evolution of the species, as described by Charles Darwin and by modern genetics. Evolution can indeed be seen as an optimization mechanism, which finds the best fitted set of traits for a species to survive in a given environment. In this thesis, we develop a particular instance of an evolutionary algorithm, in order to solve the gene expression data biclustering problem.

The algorithm developed in this thesis is based on two main foundations. The first is the δ-bicluster model, a mathematical specification of the gene expression data biclustering problem, used by the scientists Y. Cheng and G. M. Church as a basis for their algorithmic solution to this problem. The second is the pair of SEBI and SMOB evolutionary algorithms, proposed by F. Divina, to solve the gene expression data biclustering problem using evolutionary computation, according to Cheng and Church's specification.
The main objective of this thesis was to improve the evolutionary approach used in the SEBI and SMOB algorithms, in order to be able to individuate better solutions to the gene expression data biclustering problem, according to Cheng and Church's specification. The proposed evolutionary approach should allow to deal more efficiently with the need to find several different biclusters within the data, and with the need to combine several antagonist criteria to evaluate the quality of the biclusters.
In particular, we were suggested by Professor Divina to incorporate in our algorithm a new proposal of a generic evolutionary algorithm component, called probabilistic encoding. The genericness of this component holds in the fact that its interest is not limited to the gene expression data biclustering problem. On the contrary, it could be potentially incorporated in many instances of evolutionary algorithms to improve their efficiency, for a wide range of different problems to solve. A secondary objective of this thesis was thus to propose a first test of the probabilistic encoding evolutionary technique on a life-size problem.

The result of the work produced in this thesis is the MOBPEOC biclustering evolutionary approach, for Multi-Objective Biclustering with Probabilistic Encoding and Overlapping Control. The MOBPEOC approach was developed by coupling both a theoretical and experimental process. In the theoretical process, we reviewed many existing techniques presented in the literature, which, coupled with probabilistic encoding, would be liable to improve the efficiency of our biclustering mechanism. In the experimental process, the efficiency of the proposed techniques, coupled with probabilistic encoding, was tested, compared and improved by applying them to real biological data, already analyzed using the SEBI/SMOB and other biclustering algorithms.

In this report, we present the context and put forward the architecture and results of the MOBPEOC approach. The text is divided in two main parts. The first part, involving chapters 1 and 2, consists in a compilation of the current state-of-the-art, where we expose the theoretical context and the existing practical work in which the MOBPEOC approach takes place. The second part, involving chapters 3 and 4, is a detailed presentation of the work achieved in this thesis, where we present the algorithmic structure and the experimental evaluation of the MOBPEOC approach.
In chapter 1, we define what optimization problems are, and we show how optimization techniques simulating models of the natural evolution of the species can be implemented. We concentrate on the particular class of genetic algorithms, the variant of evolutionary algorithms implemented by the SEBI and SMOB algorithms, and also within the

MOBPEOC approach. First, we present a framework encompassing the elemental components of a genetic algorithm. Then, we present some more advanced components that could allow to deal more efficiently with the complex aspects of the biclustering optimization problem. Finally, we propose a brief evaluation of evolutionary computation as an optimization technique.

In chapter 2, we present the gene expression data biclustering problem, and detail the relevant existing work on this topic. First we detail the δ-bicluster model specification of the gene expression data biclustering problem, review its algorithmic complexity, and present the algorithmic solution presented by Cheng and Church. The results obtained by Cheng and Church on two real biological datasets (known as the Yeast and Human datasets) are also presented. Then we establish how the δ-bicluster model specification of the biclustering problem can be expressed as an optimization problem, and demonstrate why evolutionary computation is a particularly appropriate technique to solve it. Afterwards, we detail the algorithmic structure of the SEBI and SMOB genetic algorithms and compare their results on the Yeast and Human datasets with the ones obtained by Cheng and Church. Finally, we briefly review the other existing evolutionary approaches to the gene expression data biclustering problem.

In chapter 3, we detail the algorithmic structure of the MOBPEOC approach. We present the different techniques that were combined to create the MOBPEOC algorithm, and justify how they can be useful to offer a better biclustering efficiency. We notably present the probabilistic encoding technique introduced with MOBPEOC, and make many references to techniques presented in chapter 1. A systematic description of the MOBPEOC algorithmic process and of the main parameters within the method is also proposed.

Finally, in chapter 4, we expose the results obtained by MOBPEOC on the Yeast and Human biological datasets. For each dataset, the chosen parameters values are detailed and justified. Then the obtained results are presented, commented and compared with the ones obtained using the SEBI/SMOB and the Cheng and Church's algorithm.

Part II

Evolutionary computation and biclustering of gene expression data, a state of the art

Chapter 1

Genetic algorithms and optimization

Contents

1.1 Evolutionary computation and genetic algorithms: an historical introduction

Neo-darwinism [Back et al., 1999, Chapter 1, 4-5] [Dréo et al., 2003, Page 69] refers to the modern synthesis between Darwin's evolution theory and current knowledge in the field of genetics. By now, it is the only available scientific theory of species evolution that has not been invalidated by observations. It explains the wide variety of species on Earth by the observed fact that species continuously adapt themselves to their environment and its changes. Synthetically, Darwin's theory states that evolution in a group of individuals from the same species occurs from *competition* for survival and reproduction between individuals. This involves a *selection* of individuals best fitted to their environment. These individuals have statistically more chances to have offspring and to transmit those traits that gave them a competitive advantage for survival and reproduction. Advantageous traits tend thus to generalize across the generations, while disadvantageous ones tend to disappear. Genetics allows to explain how traits can be transmitted to offspring through *reproduction*, and how new and potentially advantageous traits can appear in the group through genetic *recombination* and *mutation*.

The history of what is now called evolutionary computation (EC) [Back et al., 1999, Chapter 6] [Dréo et al., 2003, Pages 69-70] is usually said to have begun during the mid-1950s, when the first computer models of the natural process of species evolution, as described by the neo-darwinian theory, appeared. The purpose of this work was not only to give a better understanding of evolution and adaptation through computer simulation, it was also to use evolution-like mechanisms as a model for programming a computer so that it could discover the solutions best fitted to a problem.
The neo-darwinian mechanism of evolution can indeed be seen as a powerful and robust optimization process. Evolution has been seen as continuously adapting species to an incredible variety of environments that can sometimes be frequently changing and very hostile. This gave birth to the huge amount of various and surprising species that can be found almost everywhere on the Earth. It seems thus particularly interesting to use computer simulated parts of this process to solve optimization problems in general, as they can be found notably in science, technics and economics.

As time went by, three approaches of EC stood out: evolution strategies (ESs, I. Rechenberg, 1965 [Rechenberg, 1965, Beyer, 2001]), evolutionary programing (EP, L. Fogel, 1966 [Fogel et al., 1966, Fogel, 1999]) and genetic algorithms (GAs, J.H. Holland, 1967 [Holland, 1975]). Although the bases of what are now considered as the three main branches of EC were all established by the mid-1960's, they were developed quite independently, ignoring each other for about 25 years. During this time, the field received little attention from the scientific and engineering communities, with only about a few hundreds of publications in 20 years.
It is only at the end of the 1980's than the field really started gaining popularity. In 1985, the first International Conference on Genetic Algortithms (ICGA) was hold in Pittsburgh, Pennsylvania. In 1989, D. Goldberg published a book titled "Genetic Algorithms in Search, Optimization and Machine Learning" [Goldberg, 1989] in which he enlightened in a very operational way the theory behind GAs and their applications. In the following years, this book is said to have been an important catalyst that helped to make GAs very popular to a more general audience of scientists and engineers. In 1991, the first Parallel

Problem Solving from Nature (PPSN) conference was hold by the ESs community, which lead to a first interaction between the GAs and ESs communities, and then to the creation of a scientific journal for the field. This journal included the EP community as well, as they organized their first conference in 1992, and its first issue was published in 1993. The name of the journal was "Evolutionary Computation", and it became the name by which the research field comprising the three subdomains is usually referred.

At present, evolutionary computation research and engineering in general have grown substantially, thanks to cross-fertilization between ESs, EP ang GAs, and through the large number of various applications where they have been applied. This growth is also both a cause and a consequence of the many events organized, and of the increasing number of books and articles written in the field.

By now, there is not one unified theory of evolutionary computation [Back et al., 1999, Chapter 7]. An evolutionary algorithm can be defined as an optimization algorithm that uses a mechanism inspired by neo-darwinian evolution. Evolutionary computation techniques constitute then a large toolbox to build evolutionary algorithms. New tools and ways of combining them are created, and existing ones are studied and improved. As there is no encompassing simple theory, developing an evolutionary algorithm for a given optimization problem is more or less a trial and error adaptation process. This process is driven by a mix of field knowledge and creativity, and verified by a theoretical and experimental analysis.

In this thesis, we develop an evolutionary algorithm in order to solve a hard biology-related optimization problem: biclustering of gene expression data. If many hybridizations occurred, the three original approaches of evolutionary computation, evolution strategies, evolutionary programing and genetic algorithms, are still often used as a basis to describe the mainstream different instances of evolutionary algorithms. Our work will essentially be based on the Goldberg's original instantiation of genetic algorithms. First, because the algorithm developed here pushes existing work based on genetic algorithms further, in order to solve the gene expression data biclustering problem. Secondly, we will use existing extensions of classical GAs, developed to solve particular classes of optimization problems, which encompass the biclustering of gene expression data. Finally, and maybe most importantly, we use our particular implementation of a genetic algorithm to introduce a new evolutionary technique that extends the traditional genetic algorithms-originated binary encoding: probabilistic encoding.

In this chapter, we introduce the state-of-the-art conceptual elements necessary to understand and evaluate the genetic algorithm implementation we developed and the probabilistic encoding technique we propose. As we begin section 1.2 of this chapter, we will present a general definition and establish a vocabulary, scope and taxonomy for optimization problems. In the context of solving classical combinatorial optimization problems, we will then propose, in section 1.3, an extensible framework that encompasses the classical Golberg's genetic algorithm structure and concepts, and its common variants. Biclustering of gene expression data is a particularly complex optimization problem, as it can be seen as a multimodal and multi-objective problem. As a consequence, we review the intrinsic qualities of our framework and detail its commonly proposed extensions to solve such specific classes of optimization problems. Multimodal optimization problems are treated in section 1.4, and multi-objective optimization problems in section 1.5. We finish this chapter with a general evaluation of evolutionary computation techniques as opti-

mization problem solvers (section 1.6).

This chapter is notably inspired by the following books on genetic algorithms and evolutionary computation: "Genetic Algorithms in Search, Optimization and Machine Learning" [Goldberg, 1989], "Evolutionary Computation 1: Basic Algorithms and Operators" [Back et al., 1999], "Evolutionary computation 2: advanced algorithms and operators" [Bäck et al., 2000], "Introduction to evolutionary computing" [Eiben and Smith, 2003], and "Métaheuristiques pour l'optimisation difficile" (French) [Dréo et al., 2003]. In order to make the text more readable, references to these books will not be repeated in the next sections.

1.2 Optimization problems

1.2.1 Definition, vocabulary and scope

Optimization is a very common concept, which is used to describe many various problems in many various contexts. As a starting point for this section, we propose thus a general and formal definition for optimization.

Definition 1.1. *An **optimization problem** is a problem that determines explicitly or implicitly the set of potential solutions to a problem and defines one or several criteria that can measure the quality of each particular solution. Solving the problem means to extract solutions from the set that have the best or a sufficiently high quality in the set.*

In order to solve an optimization problem with a computer, it should be possible to give a formal definition of the set of potential solutions, and of the quality criteria, so that the set can be explored and the quality of the solutions can be evaluated by the computer.
It should be noted that in order for such a formalization to be possible, the problem must often be simplified, especially in the case of informal and complex problems.

When a formal definition of the set of solutions is given, it is called the *search space* S. The search space can be defined in an extensional way (i.e. by listing all the solutions in the set) or, more frequently, in an intensional way (i.e. by listing the necessary and sufficient conditions for elements to be part of this set). In this last case, programming a computer to explore the set, i.e. to build new solutions that fulfills the conditions, can be a challenging task if the conditions are complex.

In the most general case, each quality criterion is modeled by a real-valued function, that associates each element of the search space with a real number measuring its quality. Such a function is called an *objective function* $o : S \to \mathbb{R}$.
Measuring the quality using an objective function can be done in two ways. The greater the real number is, the better the solution is, or the other way round. Solving the problem means thus to find solutions that respectively maximize or minimize the objective function(s). As the problem of minimizing $o : S \to [-\infty, +\infty[$ can always be transformed in the equivalent problem of maximizing $o' : S \to [-\infty, +\infty[, o' = -o$ and vice-versa, from now on, we will only consider maximization problems.
A real-valued function $o' : S \to [-\infty, +\infty[$ can always be transformed into a positive real-valued one $o'' : S \to [0, +\infty[$, such that:

$$
\begin{array}{rl}
 & \forall s,t \in S, \\
 & o'(s) < o'(t) \Leftrightarrow o''(s) < o''(t) \\
\wedge & o'(s) = o'(t) \Leftrightarrow o''(s) = o''(t)
\end{array}
$$

For example, such o" could be:

$$
o'' : S \rightarrow [0, +\infty[, o''(s) = \begin{cases} o'(s) + 1 & if\ o'(s) \geqslant 0 \\ -\frac{1}{o'(s)-1} & if\ o'(s) < 0 \end{cases}
$$

We can thus, without loss of generality, work only with positive real-valued objective functions to be maximized.
If there are several different quality criteria that have to be used simultaneously to evaluate a solution, there will be several objective functions corresponding respectively to each criterion. Optimization problems with only one objective function are called *single-objective*, and optimization problems with more than one objective function are called *multi-objective*. Usually multi-objective optimization is treated as a separate problem, and optimization often refers only to single-objective optimization. We will discuss multi-objective optimization and how genetic algorithms can solve multi-objective problems in section 1.5. For now, we will only consider optimization as single-objective optimization.
Nothing guarantees that the objective function of the problem cannot give the same measure to several different solutions. Thus there can be several different, but quality-wise equivalent, best solutions in the set. They are called the *optimal* solutions or *optima*. If some sufficiently good but not optimal solutions can be accepted, it typically means that all the solutions whose quality measure is above a given threshold, or sufficiently close to the quality measure of the best solutions (these are called sub-optimal solutions) can be accepted. As a consequence, there can be several solutions in the set that are candidates to solve the problem. The adopted policy to choose which of these solution(s) should be searched for is typically a function of the context of the problem. Two extreme cases are *unimodal* and *multimodal* problems. In unimodal problems, it is known that there is a single solution that outperforms all the others, and it has to be found. In multimodal problems, the problem is known to have several interesting different solutions, and the largest possible number of different optimal, sub-optimal, and sometimes even sufficiently good solutions should be found. We will discuss multimodal optimization and how genetic algorithms can solve multimodal problems in section 1.4. For now we will only consider the simple case of unimodal optimization.

Taking the introduced formalization and simplifications into account, we can now give the common operational definition of optimization [Barichard, 2003, Page 8] that will be used as a basis to introduce the genetic algorithms framework in the next section.

Definition 1.2 (Simple and operational optimization problem (unimodal - maximization))**.**

Let S be a set of solutions to a problem, and let $o : S \rightarrow [0, +\infty[$ be an objective function that measures the quality of these solutions,
Then find the optimal solution $m \in S \mid \forall s \in S, o(m) \geqslant o(s)$

The search space S can be a finite, countable or uncountable set. In the two first cases, we talk about *discrete* optimization, and in the last one, of *continuous* optimization. Genetic algorithms are typically made for finite search spaces, but work has been done to adapt them to general discrete (see, for example, [Beyer et al., 2002]) and continuous problems (see, for example, [Chelouah and Siarry, 2000]). Note that the easiest way to solve problems where the search space is equipotent to the integer or to the real numbers set, is to render the search space finite using the finite sets of traditional binary machine representation of integer and real numbers, or equivalent representations. In this work, we will only consider optimization problems with a finite search space.

1.2.2 Combinatorial optimization problems and evolutionary algorithms

For a finite unimodal optimization problem [Hao et al., 1999], the basic algorithmic solution typically assesses exhaustively as many solutions as needed in the search space, in order to be able to prove formally that a given solution is at least better than any other solution in the search space. This solution is the optimum returned by the algorithm. This is the implicit and underlying principle of *"exact" methods*. In hard problems, exact methods may have to evaluate exhaustively all the solutions of the search space, or at least an important part of them, in order to find the optimum. In this case, exact methods cannot cope with problems where the search space is too large for such an exhaustive search to be achieved in a reasonable time.
This typically happens for problems defined by a finite number of elements to be combined in the most effective way. The search space is thus the set of allowed combinations between elements. If the problem is NP-Complete, i.e. it can be conjectured that there is no exact algorithm capable of finding the best combination in a polynomial time of the number of elements to combine, then for a sufficiently large number of these elements, the number of potential combinations is huge, and the time needed by an exact algorithm to find the best one becomes unreasonable. Such problems are called *combinatorial problems* (note that some authors uses combinatorial as a synonym of discrete, and others for all problems with a huge but finite search space).

To solve combinatorial problems, *heuristic methods* are typically used. These methods do not rely anymore on a search process that formally proves that a solution is better than any other one in the search space, but use some particular "rule of thumb" to try to find the optimum, among the set of possible solutions. On the one hand, this involves that they typically lose the property of always finding the optimal solution. On the other hand, they usually can find very good solutions by exploring only a small part of the search space.
Evolutionary algorithms are one of these heuristic methods. The "rule of thumb" used by evolutionary computation is to be found in the parallel between the elements to be combined in a combinatorial optimization problem, and the traits that combine themselves to create an individual in neo-darwinian evolution. In applying an evolution-like process to a population composed of a small and limited number of potential solutions of a combinatorial optimization problem, one hopes that advantageous elements and advantageous combinations of elements will emerge in the population as generations pass.

1.2.3 Examples of optimization problems

The scope of optimization problems as described in definition 1.1 is very large. Many problems in science, technics, engineering and in everyday-life can be solved as optimization problems (see, for example, [Neumaier, 2010b]). As a result of the variety and complexity of these problems, there is a lot of research and engineering activity to develop, improve and use a wide panel of optimization techniques (see, for example, [Neumaier, 2010a]). In the following lines, we present two simple examples of optimization problems, and illustrate how they fit to the theoretical definition and taxonomy given in the previous paragraph.

A Example #1: A continuous problem

Definition 1.3 (Box with square base problem)**.** *A parallelepiped box of height h has a square base of side x. The box is made of plastic, which costs 5 gold per square unit. Given that the volume of the box must be 100 volume units, find the value of x for which the box will be the cheapest to make.*

Search space definition	The set of all possible side lengths, i.e. $[0, +\infty[$.
Objective function(s) definition(s)	The objective function must associate each possible side length with the cost of the resulting box. The cost c can be expressed as a unique function of x in the following way: $c : [0, +\infty[\rightarrow [0, +\infty[,\ c(x) = 5 * \text{Surface of the box}$ $= 5 * (2 * x^2 + 4 * x * h)$ $= 5 * (2 * x^2 + 4 * x * h * \frac{x}{x})$ $= 5 * (2 * x^2 + \frac{4*(x^2*h)}{x})$ $= 5 * (2 * x^2 + \frac{4*\text{Volume}}{x})$ $= 5 * (2 * x^2 + \frac{400}{x})$ $= 10 * x^2 + \frac{2000}{x}$ This is a positive real-valued function, which has, in this case, to be minimized.
Number and nature of quality criterion(s)	The problem is single-objective, the objective function being the cos function c.
Quality of solution(s) searched for	It depends on the problem context. We can suppose for example that we only want to find the or all the optimal solutions.
Nature of search space	The search space is an infinite uncountable set. This problem is a continuous problem.

B Example #2: A combinatorial problem

Definition 1.4 (Traveling salesman problem (TSP))**.** *Given a list of cities to be visited, find the shortest possible tour that visits each city exactly once.*

Search space definition	The set of all possible tours that visit each of the cities exactly once.

Objective function(s) definition(s)	A positive real-valued function that associates to a tour its total length (to be minimized).
Number and nature of quality criterion(s)	The problem is single-objective. But other objectives could be easily added. For example, if the tour is made by car, the total cost, including fuel and toll, could be minimized at the same time as the length of the tour.
Quality of solution(s) searched for	It depends on the problem context. Nevertheless, one could imagine that all the existing shortest and nearly shortest tours could be interesting and should be searched for. In that particular case, we would be facing a multi-modal problem.
Nature of search space	The search space is a finite set. This problem is a combinatorial problem. Tours can be seen as a kind of combination between cities, and the number of possible tours grows exponentially as the number of cities increases (#tours = #cities!). This problem is a well-known NP-Complete problem [Garey and Johnson, 1990].

1.3 A framework for common genetic algorithms

In this section, we propose an extensible framework that encompasses the concepts and structure of the classical genetic algorithm, defined by D. Goldberg in [Goldberg, 1989], and of its common variants, and we show how this framework can be used as an heuristic method to solve the simple optimization problem specified in definition 1.2, supposing a finite search space, and typically considering combinatorial optimization problems.
In subsection 1.3.1, we propose the simple evolution model, inspired by the neo-darwinian evolution model, that the genetic algorithms will typically implement. In subsection 1.3.2, we show how this model is translated into the general algorithmic procedure of genetic algorithms. Finally, in subsection 1.3.3, we describe how this general algorithmic procedure is typically implemented into a functional genetic algorithm.

1.3.1 Model of evolution

Evolutionary computation is inspired by the neo-darwinist model of the evolution process to propose an evolution-like optimization procedure. As a first glance at the layout, concepts and vocabulary genetic algorithms, we describe the simple computer model of evolution implemented by GAs. In this model, one can find the classical vocabulary of neo-darwinism, and a mechanism that looks like a very simplified view of the neo-darwinist evolution model. But it should be noted that the reality of evolution is much more complex and different of what is stated here.

Any living creature owns a set of *genes*. All the genes of this set are grouped in one or more *chromosomes*, which are thus sets of genes. Each gene has a fixed position in its chromosome called its *locus*. There is a finite number of possible variants for a same gene. Variants are called the *alleles* of the gene, and each living creature owns, for each gene, one of its possible alleles. The members of a same *species* share the same set of genes, organized in a same set of chromosomes. Different individuals of the species only differ

by the allele values of their genes. The set of alleles of an individual is called its *genotype*. The genotype is in fact the "building plan" of the individual and determines thus the full set of observable physical characteristics, called *traits*, of the individual. This last set is called the *phenotype*. Different individuals in the species will have different phenotypes, i.e. physical characteristics, because they have different genotypes, i.e. different sets of alleles.

Two individuals of the same species, called parents, can *reproduce* to create new members of the species (with thus the same set of genes and chromosomes), called *offspring*. The genotype of an offspring takes for each gene, either the allele value of the mother or the allele value of the father. He will thus typically inherit some traits from each of its parents.

A *population* is a group of living creatures of the same species who share a same environment. The members of a population are isolated from other living members of the species, so that reproduction can only occur inside the population. Members of the population are called *individuals* and individuals are in competition for reproduction and survival. Some individuals in the population may have particular traits that make them better adapted to this environment than the other ones. These traits will thus give them more chances than the other individuals to reproduce, and to survive in the environment long enough for having the opportunity to reproduce. These individuals will thus have more chances to transmit to the next *generation* parts of their genotypes and thus potentially the interesting traits that gave them the advantage over the other individuals. As the generations pass, the frequency of disadvantageous alleles will decrease and the disadvantageous traits will tend to disappear, while the frequency of advantageous alleles will increase, and the advantageous traits will spread. This mechanism is called *natural selection*.

If offspring inherit traits from their parents, their genotype is nevertheless different from those of their parents. The genotypic *recombination* that occurs during reproduction is one of two sources of genotypic variety. The other one is *genetic mutation*. Any living creature can sustain genetic mutations. A genetic mutation changes the allele value of a gene of the creature that sustains it. In a population, genetic mutation can thus introduce allele values that did not exist for a particular gene in the individuals composing the population.

These two sources of genotypic variety allow the appearance of new inexistent traits or combinations of traits in the population, that can be advantageous, and offer new chances of environmental adaptation to the population as the generations pass.

A typical genetic algorithm simulates this model with a computer in order to solve an optimization problem. The idea is to apply the mechanism described above not to a population composed of individuals from a given species, but to a small population of potential solutions from the search space of the optimization problem. For optimization to occur, we replace the criterion of adaptation to the environment by a criterion of "adaptation to the problem", i.e. of quality, measured by the objective function of the problem. The basic analogy that sustains evolutionary computation is this one: the species is the search space, and the environment the quality criterion. We can thus hope that, as the generations pass, these characteristics that make solutions better will emerge, producing in that way solutions very well adapted to the problem. In the next section, we show how GAs simulate such a process through an algorithmic procedure.

1.3.2 Main algorithmic procedure

A The generations loop

The core mechanism of a genetic algorithm is an iterative process, called the *generations loop*. This iterative process repersents the passing of time and generations in the evolution model. The generations loop creates a sequence of populations. A population here has a different meaning than in the evolution model:

Definition 1.5. *A **population** is multiset of elements from the search space of the optimization problem.*

Each iteration of the generations loop works on one population, that contains all the living individuals at the time of this iteration. Such a population is called a generation and the number of individuals in a generation is typically very small compared to the size of the search space. The processing done by an iteration of the loop is to create the generation of the next iteration, from the current one. Between two generations, new individuals will appear, as some individuals of the current generation will reproduce, and some of the individuals of the current generation will die and disappear. Moreover, genetic mutations can also occur on some individuals. We can thus give a first pseudo-code description of the general algorithmic procedure of GAs:

```
Population currentGeneration,nextGeneration;
while (Optimum_is_not_found) {
        nextGeneration    = createFrom(currentGeneration);
        currentGeneration = nextGeneration;
}
```

We will describe more precisely how the "createFrom" procedure works in the next paragraph. But first, we can notice that we are already facing two questions:

1. How the algorithm is started, i.e. how the first current generation is created?
2. How the algorithm is stopped, i.e. how do we know that the optimum is found?

For the first question, it appears that the individuals composing the first generation must be picked in some way from the search space. Genetic algorithms are not an exhaustive search optimization method: they start from a first population and try to evolve it towards optimality across the generations. It appears thus that the initialization of the first generation can affect the convergence speed, and potentially the final solution returned. Usually, if one knows information about the characteristics of the optimal solutions to find, one will populate the first generation with individuals that have these characteristics. Otherwise, individuals are picked randomly out of the search space. Research has been done on more sophisticated techniques for algorithm initialization (see for example [Rahnamayan et al., 2007]).
For the second question, if the value of the objective function at optimum is known, one could stop the algorithm as soon as an individual giving this value to the objective function is found, but this value is not always known. Moreover, the algorithm could take a very long time to find the optimum, and as GAs are not an exact method, they typically offer even no guarantee to find the optimum. Typically, a trial and error approach is used.

A termination condition is first defined. For example, if no optimum has been found, the algorithm is stopped after a fixed number of iterations of the generations loop. Then the results and some statistics on the population evolution during the run are used to evaluate and eventually modify the chosen number of iterations. The process is then repeated to adjust this maximum number of iterations, which is thus the first parameter we introduce in the algorithm. The returned final solution will be typically the best individual of the last generation, or of the whole set of successive generations.

Taking these two answers into account, we can already refine our pseudo-code description of the algorithmic procedure of GAs:

```
// ———— Parameters ————
maxNumberOfGenerations = ... ;
// ———— Parameters ————

Population currentGeneration,nextGeneration;

currentGeneration = pickIndividualsFrom(searchSpace);
numberOfGenerations = 0;

while (Optimum_is_not_found || numberOfGenerations<maxNumberOfGenerations) {
        nextGeneration    = createFrom(currentGeneration);
        currentGeneration = nextGeneration;
        numberOfGenerations = numberOfGenerations + 1;
}

return currentGeneration.bestIndividual();
```

B Genetic operators, fitness and genotype

B.1 The general selection scheme. When creating the next generation from the current one, the algorithm will mimic what happens in the evolution model we presented, so that natural selection and genotypic variety may be able to make the population evolve in the direction of a better adaptation to the problem. This is achieved through applying a set of *genetic operators* to build the new generation from the current one.

Definition 1.6. *A **genetic operator** is a partially random mechanism, that receives a population as input, and produces a population as output. Elements of the input population are called parents, and elements of the output population are called offspring. The parents set and its content are not modified by the operator.*

The defined operators, the way they proceed and the way they are combined to create the next generation from the current one are called the *general selection scheme* of the algorithm. If all evolutionary algorithms use a generations loop defining a sequence of populations, each family of EAs (ESs, EP and GAs), and their hybrid variants, differ by the general selection scheme used. We describe now the general selection scheme of traditional GAs.

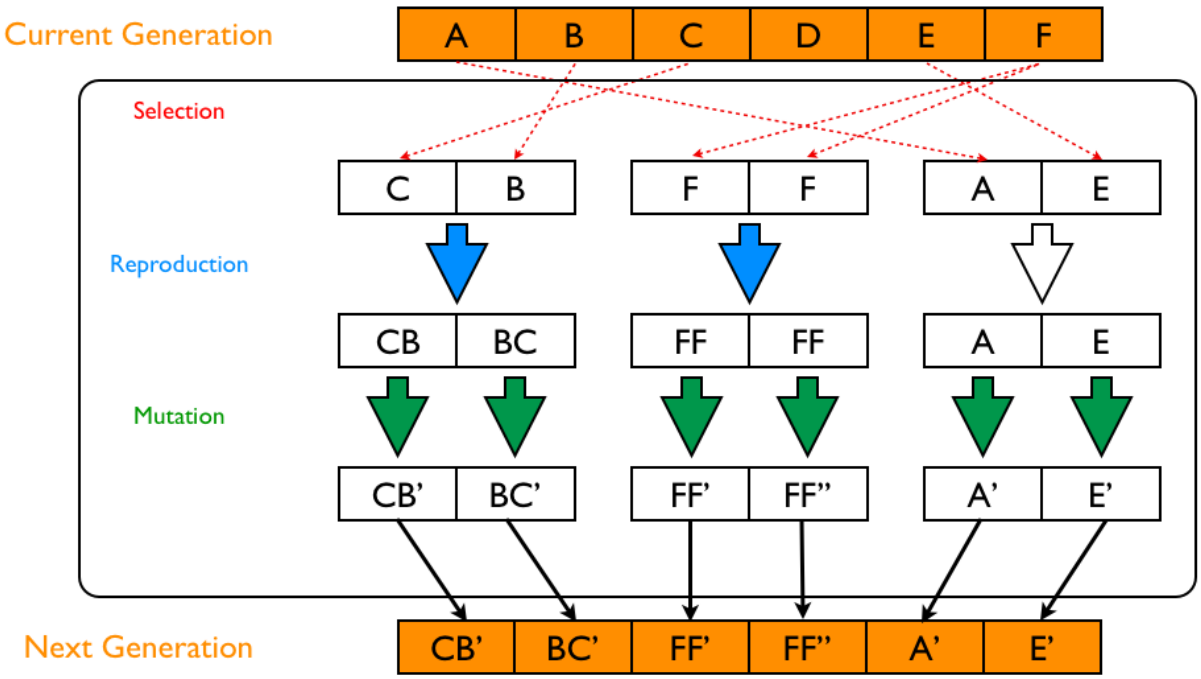
In a GA, three types of genetic operators are typically used: a selection operator, and two variation operators, a reproduction operator and a mutation operator.

The selection operator is used to select an individual from the current generation. The individuals in the current population will have more chances to be selected if they are better adapted to the problem than the other ones. At the beginning of an iteration of the generations loop, the selection operator is used twice independently to get a couple of individuals from the current generation. This couple will be reproduced or not, based on a fixed probability rate of reproduction. If they are not reproduced, both individuals are said to survive and will be copied to the next generation, after sustaining mutation. If they are reproduced, two offspring are produced (a popular variant with only one produced offspring also exists) and they will be copied to the next generation, after sustaining mutation. A reproduction operator creates thus two offspring from two parents, and a mutation operator one offspring from one parent.

In GAs, the number of individuals in a generation is supposed to stay constant at each iteration of the generations loop. The process is thus repeated and individuals copied to the next generation until the new generation contains the required number of individuals. The algorithm switches then to the next iteration of the generations loop. Remark that nothing prevents, in such a process, a same individual from the current generation to be selected several times by the selection operator.

This mechanism introduces two new parameters in the algorithm: the reproduction probability rate and the number of individuals in a generation. In the following parts of this chapter, we will often refer to the number of individuals in a generation parameter using the notation N_{gen}.

The following figure illustrates the creation of a new generation during an iteration of the generations loop. The current generation is composed of six individuals A, B, C, D, E and F. The selection-reproduction-mutation process is used three times to create the six individuals of the next generation CB', BC', FF', FF", A' and E'. A red arrow corresponds to a selection operator, a blue one to a reproduction operator, and a green one to a mutation operator.



Remark that in this example individual D is never selected, while individual F is selected twice. Moreover, notice that individuals A and E are not reproduced, but copied to the next generation directly after sustaining mutation.
We detail now the three types of operators, and go further in our description of the structure of genetic algorithms.

B.2 Selection operator and fitness. A selection operator receives as input the current generation and returns as output one individual in this generation. As in the evolution model, selection will essentially work at random, but the best adapted individuals in the generation will have statistically more chance to be selected.
The selection operator will thus have to compare, inside a generation, the individuals' adaptation to the "environment", which is the optimization problem. The comparison will thus be based on the objective function of the problem. If the objective function returns the absolute quality measurement of an individual in the whole search space, which has a meaning in the context of the problem, such a measurement may not be convenient to compare the small number of individuals that compose the current generation, at each iteration of the generations loop. For example, it may take a lot of time to be computed, so that using an approximation could be sufficient for comparison. It may also discriminate too much or not enough between the individuals in the generation, so that selection does not work in an effective way. In some cases, one wants to add penalties or rewards to solutions exposing a given set of traits in order to drive the search process. One can even want to establish the adaptation value of one individual in function of the composition of the whole population. For these kinds of reasons, a distinction must be made between the objective function, given by the problem, and the actual function used by the GA to measure the adaptation of individuals in one population, called the *fitness function*. Depending on the problem and on the implementation choices for the selection operator, creating a good fitness function from the objective function can be a challenging task.

B.3 Variation operators and genotype. Reproduction and mutation are the two variation operators, and they mimic the corresponding process in the evolution model. Reproduction receives a population of two parents et produces a population of two offspring whose genotype are random mixings of the parents' genotypes. Mutation receives one parent and produces one offspring whose genotype is a random variation of the parent genotype.
In GAs, phenotype will be the abstract element of the search space of the optimization problem, and genotype will be the computer data structure representation used to encode such an element, and which the GA will manipulate. The choice of such a coding will depend on the nature of the elements in the search space, and influence the implementation of the variation operators and their efficiency in driving a powerful evolution process.

C Data structures

In the implementation of a GA, the elements of the search space, often called the *phenotypic space*, are described and manipulated using only two values: the fitness value, and the genotypic representation. We can give a more formal definition of how these two important values are calculated.

Definition 1.7. *The **fitness function** f of the GA associates, to each element of the search space in a given population, a fitness value derived from the objective function, that is used to measure the relative adaptation of the element to the problem, compared to other elements in the population.*

Definition 1.8. *Let G be the set of data structures chosen to represent elements of the search space within the GA, called the* genotypic space*. The **genotypic encoding function** of the GA is a total injective function $c : S \to G$ that associates each element in the search space with its particular representation as a data structure.*

Fitness contains all the phenotypic information about a search space element, and is the only information the selection operator typically takes care of. Genotypic representation, or genotype, contains all the genotypic information about a search space element and is the only information variation operators take care of.
Typically, when the first generation is initialized, a multiset of genotypes is generated. New genotypes are then created by variation operators at each iteration of the generations loop. The fitness function value F of a genotype g in a generation P will only be calculated on demand when the genotype g is used as parent of a selection operator.

D The GA algorithmic procedure

We finally give here a more detailed pseudo-code description of the general genetic algorithm procedure, taking into account the information given in this section. In the next section, we show how this algorithmic procedure can be implemented into a functional and effective genetic algorithm.

```
// ———— Parameters ————
maxNumberOfGenerations = ... ;
numberOfIndividualsPerGeneration = ... ;
reproductionProbabilityRate = ... ;
// ———— Parameters ————

Population currentGeneration,nextGeneration;

currentGeneration = pickIndividualsFrom(searchSpace,
                                        numberOfIndividualsPerGeneration);
numberOfGenerations = 0;

while (Optimum_is_not_found || numberOfGenerations<maxNumberOfGenerations) {

        for (int i = 1; i <= numberOfIndividualsPerGeneration/2 ; i++) {

                father = selectionOperator(currentGeneration);
                mother = selectionOperator(currentGeneration);

                if (random(0,1)<reproductionProbabilityRate) {

                        [son,daughter] = reproductionOperator(father,mother)

                        son = mutationOperator(son);
                        daughter = mutationOperator(daughter);

                        nextGeneration.add(son);
                        nextGeneration.add(daughter);
```

```
                } else {

                        newFather = mutationOperator(father);
                        newMother = mutationOperator(mother);

                        nextGeneration.add(newFather);
                        nextGeneration.add(newMother);

                }

        }

        currentGeneration = nextGeneration;
        numberOfGenerations = numberOfGenerations + 1;
}

return currentGeneration.bestIndividual();
```

1.3.3 Implementation issues and common techniques

To arrive at a fully functional genetic algorithm from the general procedural mechanism described in the previous paragraph, one should implement an efficient set of genetic operators, with appropriate fitness function and genotypic encoding. To continue with the extensible framework we are defining, we discuss (paragraphs A and B) the issues in developing good genetic operators, fitness functions, and encoding and we present the classical implementations used in GAs.
We have already defined three parameters of the algorithm, the maximum number of generations, the number of individuals in each generation N_{gen} and the rate of reproduction. An implementation will typically add many other parameters to the algorithm. Some typical examples being parameters to control the genetic operators, and parameters used to build an effective fitness function from the objective function. Parameter configuration is thus a non negligible part of GA development and we will review it in paragraph C.
From an implementation point of view, it should be remarked that the intrinsic parallelism of the mechanisms existing in the natural evolution model has notably inspired research on GAs, and many parallel implementations of GAs have been proposed so far (see, for example, [Stender, 1993, Paz, 1997]). As GAs can require a lot of computing time and memory to return interesting results on complex problem, parallel computing can be a solution to improve the performance and scalability of GAs. Some parallel GAs are not just parallel implementations of sequential GAs, as they promote new mechanisms of parallel exploration of the search space.
One should also notice that genetic algorithms and evolutionary computation in general make an intensive use of random processes. As computers are deterministic, random processes are typically implemented using pseudo-random numbers generators, which are standard features in modern programming languages.

A Implementing selection

A.1 Selection pressure and genetic drift. Selection operators are used to pick an individual in the current generation to be reproduced or directly copied to the next generation, after sustaining mutation. Selection works at random, but individuals with a better fitness should be favored. The effects of such a dual selection mechanism on the

dynamics of the evolution process can be qualitatively studied, typically using two main concepts: *selection pressure* and *genetic drift*.
Selection pressure measures how much individuals are favored as they have a better fitness in their generation. Selection pressure can be quantified through its effects. Let us consider only the selection effect in a GA, by using identity variation operators, i.e. operators whose offspring population is always the same as the parents population. At each generation, the better fitness an individual has, the more chances he will have to see its "clones" to be copied to the next generation. After a given amount of generations, clones of a same good individual from the initial population will tend to invade the full population. One can measure the chances the best individual of the initial population has to be the one whose clones will totally invade the population. One can also measure the mean required number of generations to do so. Then, the more quickly the best initial individual has chances to invade the population, the more the selection pressure will be strong.
Genetic drift is a phenomenon due to the random nature of selection. Let us use the GA with identity variation operators from above, but let us consider that the selection operators now select individuals in the current generation completely at random. One could imagine that the population would fluctuate in a total random way as generations pass. But as selection is totally random, one individual from the initial population will, at one moment, have more clones in the current population than any other one, getting thus more chances to be selected and cloned in the next population. Because of this, it can be shown that the clones of one random individual from the initial population will totally invade the population after some generations. This phenomenon is genetic drift and is even more powerful than the sampling of the current population made by the successive applications of the selection operator, independently of the individual performances, can be unfair, i.e. the variance in the number of times an individual of the current generation is selected can be high.
All other things being equal, a too strong selection pressure can thus lead to the rapid invasion, as generations pass, of the clones of an individual which may be of good quality within its generation, but of bad quality at the level of the whole search space. This is called premature convergence. On the contrary, if the selection pressure is too weak, random evolution will dominate, and convergence will be slow as good solutions could be eliminated by the genetic drift.
When clones of one individual invade the population as generations pass, because of selection pressure or genetic drift, the population is said to converge. Note that the smaller the number of individuals in each generation N_{gen} is, the more it increases the risk of an unfair sampling in favor of some individuals, and the quickness of such convergence.
The key point for selection operator design is thus the control of the global selection pressure to an efficient level so that the algorithm performs well. We review now the classical implementation techniques in GAs for selection operator.

A.2 Proportional and tournament selection. A selection operator receives as input the current generation, and produces as output one individual from this current generation. Two main classes of selection mechanism can be used to implement selection operators in common GAs : *proportional* and *tournament selection*.

A.2.1 Proportional selection

Principle Proportional selection is the original mechanism proposed by Holland and Goldberg. It gets inspired from population genetics, where the fitness of an individ-

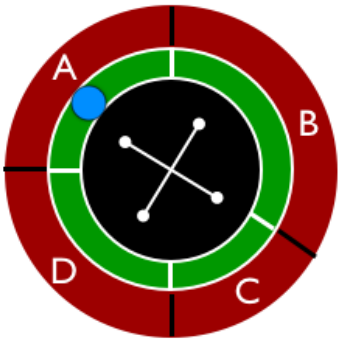
ual is defined in terms of its number of offspring. The selected individual is chosen randomly, but the chance for an individual from the current generation to be selected is proportional to its absolute quality, i.e. the value it gives to the objective function of the optimization problem.

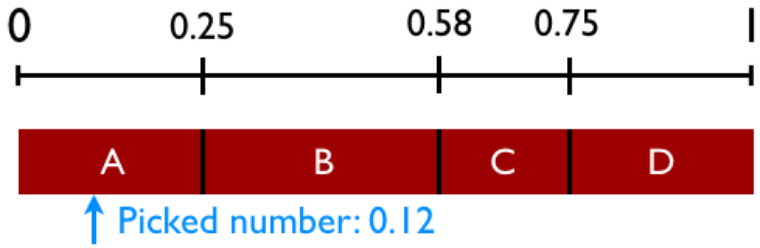
Typical implementation(s) The typical implementation is called roulette-wheel selection. Let us imagine a casino roulette wheel, containing as much pockets as there are elements in the current generation. The length of each pocket is proportional to the objective function value of the element of the current generation the pocket represents. The operator then selects one element in the current generation by throwing the ball, and choosing the element corresponding to the pocket where the ball stops.

More precisely, if o_i is the objective function value of the #i element in the generation of N_{gen} individuals, a selection probability $p_i = \frac{o_i}{\sum_{j=1}^{N_{gen}} o_j}$ is assigned at each individual i at the beginning of the iteration of the generations loop. Probabilities are then cumulated so that each individual i is assigned a subinterval of $[0, 1[$: $[\sum_{j=1}^{i-1} p_j, \sum_{j=1}^{i} p_j[$. Every time the selection operator is applied, a number is randomly chosen in [0,1[, and the operator returns the individual whose subinterval contains the picked number.

For example, with a generation composed of four individuals A, B, C and D:

Individual	o_i	$o_i/\sum o_j$
A	25	1/4
B	34	1/3
C	17	1/6
D	25	1/4





The operator would in this case select A.

Evaluation The selection pressure generated by such an operator varies with the variance of the objective function values of individuals in the current generation. If these values exhibit a strong variance, best individuals will have a very large selection probability compared to others, and they may be selected almost all the time: selection pressure will be very strong. But if they have a weak variance, all

individuals will have all almost the same probability of selection, so that selection will work quite at random, and there will not be nearly any selection pressure more.

This problem is typically solved by not using the objective function values directly as fitness for the individuals in the population. At each generation, fitness values are computed by artificially scaling the objective function values, in order to adjust the selection pressure to a given level, according to a defined policy.

Finally, as proportional selection requires selection probabilities to be calculated and scaled again for all individuals at each generation, it is considered as expensive in processing time compared to the tournament selection mechanism.

A.2.2 Tournament selection

Principle The selected individual from the current generation is the one who wins a tournament between a given number of individuals picked randomly in the current generation. The winner of the tournament is selected by taking into account the absolute quality of the participants, given by the objective function of the optimization problem.

Typical implementation(s) There are two typical ways for implementing the tournament.

In deterministic tournament, n individuals are randomly picked from the current generation, and the individual with the largest objective function value is selected.

In binary stochastic tournament, two individuals are randomly picked from the current generation, and the one with the largest objective function value is chosen with a probability $p > 0.5$.

Evaluation Tournament selection allows to control easily the selection pressure with the parameters n and p. In a deterministic tournament, as soon as the best individual of the current generation is selected for tournament, he will win the tournament. The more individuals take part to the tournament, the more the best individual has chance to be selected for tournament. The parameter n can thus be used to control selection pressure, if $n \approx N_{gen}$, then the best individual will almost always be selected by the operator, as n is decreased, selection pressure is reduced, with a minimum value of 2 for n. A deterministic tournament with n = 2 is equivalent to a binary stochastic tournament with p = 1. By decreasing p, one still decreases the selection pressure, with a minimal value of p = 0.5, where the selection pressure is zero, and the selection totally random.

Tournament selection is simple, and does only require the processing of the objective function of the individuals who take part in the tournament. Once this value has been calculated, it can be cached, and it will not be necessary to evaluate it again, if the individual participates to other tournaments in this generation or in the following ones. As no mean fitness and no scaling factors must be calculated at each generation, all the required tournaments can be lead independently from each other, paving the way for a parallel implementation.

A.3 Other general selection schemes. The general selection scheme of classical GAs uses a selection operator which picks one individual at a time, and is bases on a

sequence of independent draws with this operator for each generation. Such a mechanism can be criticized because as the draws are independent, it can happen that individuals with worst fitnesses may be selected several times while individuals with best fitnesses are never selected. By nature, this general selection mechanism can lead to sampling errors and is thus prone to genetic drift. That is said to impact the performance, even if such behavior has been seen to have a positive effect on some kinds of problems [Hancock, 1992]. Many other general selection schemes have been proposed, notably inspired by the other branches of Evolutionary Computation. A more thorough discussion of these techniques is out of the scope of this work.
We will just describe here one simple technique often used to extend the usual mechanism of GAs: *elitism*. The principle of elitism is to always keep the best individuals of the current generation in the next one. The main effect of elitism is to increase selection pressure. It has been seen to improve considerably results on some problems but tends to favor premature convergence of the algorithm on others. Elitism also allows to prevent a previously found optimal solution to be eliminated by genetic drift and guarantees that it appears in the last generation.

B Implementing variation operators and genotypic encoding

B.1 Exploring the search space. The reproduction operator receives two selected genotypes from the current generation and creates two new genotypes to be copied to the next one by randomly mixing the parents genotypes. All individuals to be copied to the new generation sustain mutation. The mutation operator randomly modifies his parent genotype to create an offspring genotype.
In the evolution model, reproduction and mutation are the only mechanisms through which new traits and new combinations of traits are proposed. Survival or not of these traits in the next generation will be a matter of chance and of the environmental advantage they can give. The same occurs in genetic algorithms. Given a set of potential solutions to the optimization problem chosen to populate the initial generation, reproduction and selection operators will be the only mechanisms that will produce new potential solutions different from the ones existing in the previous generation. Variation operators control thus how the search space can be explored by the GA. Selection operators and the general scheme will then drive this exploration.
For the GA to be effective, the production of new solutions should drive to the discovery of the optimal solution of the optimization problem, or at least to a sufficiently good one. The question is how variation operators can create new potentially better solutions at each generation. The first behavior is to produce solutions that are variations of the ones in the current generation. The hope here is that if one solution is good, some of its variations will keep what made this solution good, but will also improve the other parts, so that the result will be even better. With such a behavior, initializing the GA with a population of already good solutions is an obvious advantage. The second behavior is to produce solutions that are completely different from the ones in the current generation. The idea here is that we should not put all of our eggs in the same basket: there is no reason that some totally different solutions from the ones in the current population could not be really better or even optimal. If the first behavior is adopted, the search made by the GA will tend to be focalized to small regions of the search space, as only variations of existing solutions will be investigated, with the risk that these regions do not contain the optimum or even sufficiently good solutions. If the second behavior is adopted, the GA will be pure random search, so that the optimum can only be found by hazard. This can

take a possibly very long time, as, statistically, half of the population has to be tested for the optimum to be found in that random way.
Whether variation operators are closer to one of these two behaviors will be qualified as their *disruptive power*: identity variation operators are not disruptive at all, totally random ones are completely disruptive. Variation operators in GAs try typically to mix the two previous behaviors. Reproduction tries to randomly combine the characteristics of two interesting individuals, in order to produce an even more interesting one. Offsprings of a reproduction operator can be seen as variations of their parents, but also as quite different individuals from each of the parents taken separately. Mutation modifies an existing individual in a random way, with the hope that the new individual will be better. Mutation is important as it can introduce new individual characteristics in the population, where reproduction can typically only combine characteristics already present in the population. Mutation guarantees thus that any solution of the search space can theoretically be created by the algorithm from the solutions in the initial population, as the generations pass.
Note that variation operators work on genotypes, i.e. on the data structures that represent solutions. The phenotypic improvement offered by variation operators acting on genotypes depends thus on the genotypic encoding that maps solutions to their representation. The choice of the used representations set and of the used genotypic encoding must thus be done carefully.

B.2 Choosing a genotypic representation and designing variation operators.

The main effort in translating an optimization problem in such a form that allows to solve it with a GA is the choice of a genotypic representation and the design of variation operators. Given the search space S, one should define:

- A genotypic space G.
- A total injective encoding function $c : S \to G$.
- A set of n potentially nondeterministic reproduction operators r_i:
  $\{r_i : G \otimes G \to G \otimes G \mid \forall i \in [1, n]\,,\ \forall s, t \in S,\ \exists u, v \in S,\ r_i(c(s), c(t)) = (c(u), c(v))\}$
- A set of k potentially nondeterministic mutation operators m_j:
  $\{m_j : G \to G \mid \forall j \in [1, k]\,,\ \forall s \in S,\ \exists t \in S,\ m_j(c(s)) = c(t)\}$

The definitions of operators encompass the fact that some possible representations in the genotypic space may not map any element from the search space. According to the definitions, operators must always produce new genotypes that map elements of the search space.
One can use several reproduction operators and several mutation operators, to combine their different effects. When reproduction or mutation has to be applied, one of the corresponding operators is picked. The draw of the operator to apply can be totally random, but some operators may also be favored, and have a higher probability to be picked. New parameters pr_i and pm_j must then be introduced in the algorithm. pr_i represents the probability to pick the #i reproduction operator, with $\sum_{i=1}^{n} pr_i = 1$. pm_j represents the probability to pick the #j mutation operator, with $\sum_{j=1}^{k} pm_j = 1$.
The main guideline for defining a representation and a set of variation operators is that they must be based on the underlying optimization problem. Given the structure of the

search space and of the potential solutions it contains, data structures and operators are build to achieve an effective exploration. Of course, many representations and associated variation operators have already been defined and studied since evolutionary computation has been proposed. Moreover, they have been applied to a wide range of problems. In can thus be very interesting, while translating a problem to be solved by a GA, to review existing solutions and assess how similar problems have been solved in the past.

In the original genetic algorithms proposed by Holland, each solution is represented by a fixed-length bit string. If such a *binary encoding* can be well-suited for some kinds of problems (for example problems involving a set of boolean variables to adjust), it was proposed by Holland as a universal encoding system for all problems. Such a choice was essentially justified in the frame of a mathematical formalization of GAs centered on a "schemata theorem", and proposing an explanation of efficiency of GAs known as the building-block hypothesis. By now, this theory has been called into question [Burjorjee, 2009], and there seems to be no evidence that a binary representation must be preferred to others, whatever the underlying optimization problem is.

Nevertheless, the binary representation remains very popular, and is often considered as the canonical representation of genes associated to evolutionary algorithms. It has been successfully applied to a wide range of problems so far. Some simple classical binary variation operators have been widely used, studied and extended. For theses reasons, and because the probabilistic encoding we propose in this thesis is an extension of this binary representation, we detail in what follows these common binary variation operators.

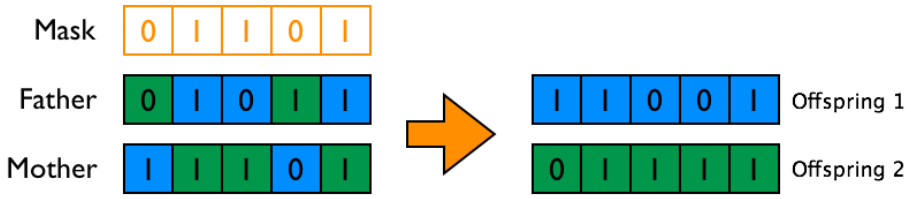
B.3 Usual binary variation operators. Three reproduction operators are typically used in practice for a binary representation: one-point crossover, two-points crossover and uniform crossover. These crossover operators are inspired by the evolution model. The two binary strings received as input by the operator are considered as the chromosomes of the father and the mother, each position in the string is the locus of a particular gene, each gene getting two alleles represented by the binary values 0 and 1. Crossover will produce two offspring whose allele value for a particular gene will be either the one of the father, or the one of the mother. One can thus describe the processing made by crossover operators as a simple procedure:

1. Every time the operator is used, create randomly a binary string, called the mask, of the same fixed length of the bit strings used to code the chromosomes.

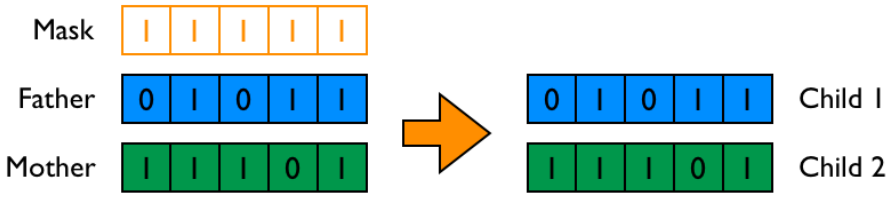
2. For the first offspring, the i^{th} value of the bit string will be the i^{th} value of the father bit string if the i^{th} value of the mask bit string is 1, or the one of the mother otherwise.

3. For the second offspring, the i^{th} value of the bit string will be the i^{th} value of the father bit string if the i^{th} value of the mask bit string is 0, or the one of the mother otherwise.

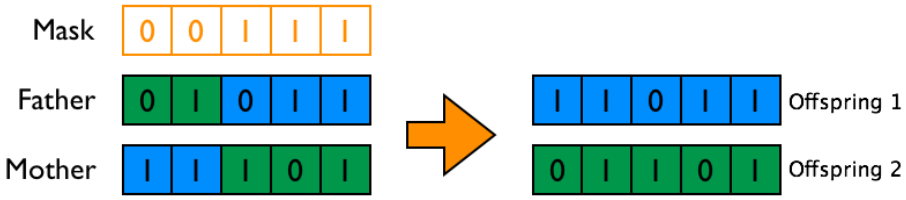
For example, for five bits-length strings:



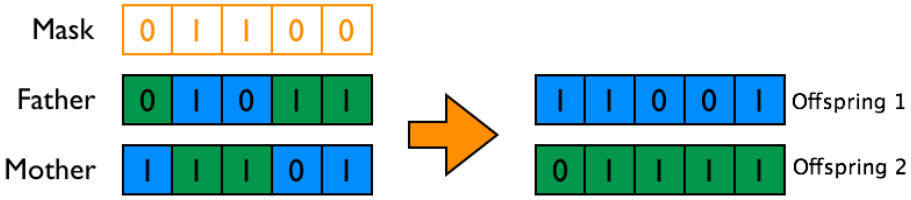
If the mask is a full 0 or a full 1 bit string, then offspring will be the same as parents:



In one-point crossover, both the mother and father string are "cut" in two parts, at the same randomly chosen position, and offspring string are made by pasting the first part of the mother string with the end part of the father string, and vice-versa. The one-point crossover operator will thus always use masks composed of l times 0 followed by m times 1 (or m times 1 followed by l times 0), with l + m = the size of the bit string, and $l, m \neq 0$:



In two-points crossover, the parents string are cut in three parts, at the same two randomly chosen positions, and the central part is switched to create the offspring string. The mask will thus be composed of three alternating sequences of 0 and 1:



One can use more cutting points. For example, in uniform crossover, the mask is randomly computed with no constraint on the number of cutting points. For each position i in the mask string, a number p is randomly picked in [0,1[, if p <0.5, the i^{th} value of the mask bit string will be 0, and it will be 1 otherwise.

In the evolution model, mutations randomly change the allele value of some genes. The mutation operator will also mimic this behavior. Every time a mutation operator is applied, it randomly switches the bit value at some positions in the string. The number

of switched bits in the string is fixed by a rate of mutation, which is a new parameter of the algorithm. Two typical implementation are used: with determinist mutation, the positions of the switched bits are picked randomly, so that the number of switched bits is the one defined by the mutation rate, and the corresponding bits are then switched. With bit-flip mutation all bits are considered one by one, and each bit can be mutated with a probability equal to the mutation rate.

B.4 Memetic or hybrid genetic algorithms. More and more recent GAs implementation tend to replace random mutation of individuals at each generation by a local optimization of individuals [Dréo et al., 2003, Page 226]. For each individual, an optimum is searched among the region of the close variations of the considered individual. This "local optimum" replaces then the considered individual. GAs that use such methods are called memetic or hybrid algorithms. As memetic computation has been developed for a few years, more complex schemas to mix local optimization with evolutionary optimization have been proposed (see notably [Krasnogor and Gustafson, 2002]).

C Parameters configuration

We have already signaled some of the common parameters that can be found in a GA:

- The maximum number of generations.
- The constant size of the population at each generation, N_{gen}.
- The rate of crossover and of mutation.
- The size of tournament in deterministic tournament or the probability of victory of the best individual in probabilistic binary tournament.
- The number of "best" individuals in the current generation that survive if elitism is applied.
- The relative rates to apply the different operators of crossover or mutation.

Many other parameters can be introduced in the algorithm as the components described in this framework are refined and extended, and other components are added. Parameters allow to tune in a very extensive way how the algorithm works, and a good parameter choice will thus be fundamental for the efficiency of the algorithm and for the quality of the returned results.

At the beginning of the 1990's, evolutionary algorithms were considered as a "robust" search method, that would exhibit a similar performance on a wide range of problems. A lot of work has been done to find a generally good parameter configuration for evolutionary algorithms [Eiben and Smith, 2003, Page 130]. Notably, studies have been lead to find an effective set of parameters able to deal with some test suites of diverse optimization problems. By now, it seems acknowledged that specific problems require a specific parameter configuration, and that the scope of an optimal set of parameters is inevitably narrow [Eiben and Smith, 2003, Page 130].

If we suppose that the structure of the algorithm has been fixed for a given problem, the typical technique used to find the optimal set of parameters for the problem is called

parameter tuning. Parameter tuning means trying to find the best parameter values for the problem through an experimental trial and error process: the algorithm is run on an instance of the problem with several sets of parameters, and the best one is chosen. The best one will generally be the one for which the algorithm returns the best solution. Typically, statistics on the evolution process are collected during the run and used to help with parameter tuning. For example, measuring the diversity in the population along the run could allow to detect some premature convergence problems.
Practically, parameter tuning often encounters many problems:

- As parameters are typically not independent, all combinations of parameter values have to be tried. Parameter tuning for a given optimization problem is itself a combinatorial optimization problem, that can be very hard to solve, if there are many wide-range parameters to tune.
- Evolutionary algorithms make intensive use of hazard (typically through pseudo-random numbers generators), and are thus non-deterministic algorithms. For each given parameter set to test, one will have to perform a statistically significant number of runs (with different random seeds for the pseudo-random number generators) with this set for the results to be compared with the other ones.
- It has been established that using a set of parameters that remains fixed during the run is not optimal [Eiben and Smith, 2003, Page 131]. Better results could typically be achieved if the parameter values were modified during the run to drive a better evolution process.

As all combinations of parameters can typically not be tested in a reasonable time, parameter tuning is often a very hasardous and time-consuming process, that often returns an acceptable but not necessarily optimal parameter set, which is fixed for the whole run.

Several approaches exist to propose solutions to these three problems. One could try to tune not an optimal set of parameter values p, but an optimal set of parameter functions $p(t)$, which returns the parameter value at each iteration t of the generations loop. This is nevertheless even harder than the simple tuning of parameter values. Mechanisms for the algorithm to automatically adapt some parameter values dynamically along the run, based on information about the past and current generations, have also been developed [Eiben and Smith, 2003, Page 131]. Finally, it has also been proposed [Eiben and Smith, 2003, Page 131] to solve the parameter tuning combinatorial optimization problem using evolutionary computation. There would be for example a GA working on a population of identical GAs optimizing the original optimization problem, each with a given set of parameters or parameters functions. One could also imagine one GA that tunes itself while solving the problem.

1.4 Extending the framework for finding diverse optimal/good solutions

1.4.1 Multimodal optimization and evolutionary computation

Multimodal optimization deals with the case where *different* optimal or sufficiently good solutions exist for the optimization problem and have to be found.

Many optimization techniques are local search techniques, i.e. they sequentially consider one potential solution at a time, jumping from one solution to another in the search space, until a solution considered as optimal is discovered or a limit of computing time is reached. In the case of multimodal problems, such algorithms will typically be run several times, in the hope that each run will return a different optimal of sufficiently good solution. Mechanisms will typically have to be used to try to prevent the algorithm to find several or every time the same solution.

Evolutionary algorithms, and genetic algorithms in particular, are population algorithms. They work on a population of potential solutions that they try to evolve to better solutions. It can thus seem natural to exploit this characteristic to find and maintain multiple different interesting solutions across the population in one run.

By now, it seems widely accepted [Singh and Deb, 2006] that reducing the selection pressure, minimizing the risk of sampling errors in applying selection, and using variation operators that are not too disruptive is not sufficient to discover and stabilize different good or optimal solutions in the population across the generations. The population will typically converge around a single solution, containing only close variations of this solution. To solve multimodal problems, new algorithmic mechanisms must be introduced in evolutionary algorithms, to promote the concurrent discovery and exploration of several interesting diverse solutions.

The first proposals on this subject were realized by Goldberg and Richardson in 1987 [Goldberg and Richardson, 1987]. Their work was based on the adaptation of ideas originating from the biological concept of ecological niches to GAs. Several different species can live in a sustainable way together in the same environment, if they are each in a different (or not too overlapping) ecological niche. An ecological niche is often described as an hyper-volume of an n-dimension space, each dimension corresponding to the available resources, such as available kinds of food, and living conditions, such as temperature, in the environment.

Goldberg and Richardson's technique was subsequently developed and tested and many different other techniques for adapting evolutionary algorithms to multimodal optimization have been proposed so far. They are referred to as *niching* methods. Niching methods typically modify the usual selection operators and general selection schemes used in evolutionary computation, so that they provide selection pressure within, but not across, different regions of similar high performing individuals individuated in the search space. These techniques can be used to extend the GAs framework proposed in the previous section, so that encompassed GAs can deal with multimodal optimization.

In this section, we first introduce the typical formal way to measure the similarity between the individuals manipulated by a GA, as it is used as a basis by most niching techniques. Then we present and evaluate sharing, the original niching technique presented by Goldberg and Richardson, whose ideas will be part of the GA developed in this thesis. In addition to the references given in the first section of this chapter, a review of most existing niching techniques can be found in the state of art made by G. Singh in [Singh and Deb, 2006], which was used as a source of inspiration for this subsection.

1.4.2 Measuring similarity between individuals

Niching methods try to let different solutions emerge in one run of the GA, by emphasizing competition within, but not across different regions of similar high-performing individuals in the search space. They should thus be able to measure how much two individuals are phenotypically similar, so that they may be part of the same region. This is typically done by making the search space a metric space, i.e. by defining a *distance* function d between any couple of solutions in the search space:

$$d : S \otimes S \rightarrow [0, +\infty[\text{ such that}$$

$\forall u, v \in S, d(u, v) = 0 \Leftrightarrow u = v$ (Identity of indiscernible solutions)

$\forall u, v \in S, d(u, v) = d(v, u)$ (Symmetry)

$\forall u, v, w \in S, d(u, w) \leqslant d(u, v) + d(v, w)$ (Triangle inequality)

The distance between two solutions should be as large as the solutions are different. The choice of a convenient distance function will depend thus on the specific solved optimization problem, and its specific potential solutions.
As GAs manipulate genotypes, the distance between the solutions represented by genotypes G_1 and G_2 will then be computed through $d(c^{-1}(G_1), c^{-1}(G_2))$ (where c represents the genotypic encoding function). An alternative solution to measure the similarity between two individuals is to define the distance directly on the genotypic space G. A distance defined on the phenotypic space (i.e. the search space S) is a *phenotypic distance*, and a distance defined on the genotypic space (i.e. the space of computer representations of the solutions) is a *genotypic distance*.
When usual genotypic representations are used in the GA, like the fixed length binary string representation described previously, some "default" genotypic distance function, measuring how two representations are different, independently of the solutions they represent, exist. With fixed length binary string representation, Hamming distance can typically be used. Hamming distance between two equal-length binary string is equal to the number of positions at which the corresponding bits in the two strings are different.

1.4.3 Fitness sharing method for genetic algorithms

The original niching method developed by Goldberg and Richardson is the *fitness sharing* (or simply *sharing*) method. Sharing acts on the fitness function used in the GA. With sharing, the fitness of an individual in the current generation equals its objective function value divided by its niche count, a quantity that is even larger than the number of other similar individuals in the generation.
Let us consider a simplistic and idealized but enlightening example: a generation of 20 individuals, where 19 individuals are very slight phenotypic variations of a same individual having a mean objective function value O_1, while the last 20^{th} individual is totally different from the others and has an objective function value of O_2. If a proportional selection mechanism is used, the probability to pick one of the 19 first individuals will be:

$$19 * \frac{\frac{O_1}{19}}{19 * \frac{O_1}{19} + \frac{O_2}{1}}$$ (fitness is the objective function divided by the niche count, which is 19 for the 19 first individuals, and 1 for the last one)

$$= \frac{O_1}{O_1 + O_2}$$

while the probability to pick the last 20^{th} individual will be:

$$\frac{\frac{O_2}{1}}{19 * \frac{O_1}{19} + \frac{O_2}{1}} = \frac{O_2}{O_1 + O_2}$$

Applying a proportional selection mechanism with sharing on the 20 individuals is thus equivalent to applying the same proportional selection mechanism without sharing on a population of two individuals, one with fitness O_1 and another with fitness O_2. The number of times one of the 19 first individuals will be selected is proportional to their shared mean fitness O_1, and the number of times the last element element will be selected is proportional to its fitness O_2. In the next generation, the number of individuals of type 1 and 2 should approximately be proportional respectively to O_1 and O_2, and the population will contain $20 * \frac{O_1}{O_1+O_2}$ individuals of the first type and $20 * \frac{O_2}{O_1+O_2}$ individuals of the second type. If sharing is applied on such a population, the scaled fitness value of an individual of first type will be:

$$\frac{O_1}{20 * \frac{O_1}{O_1+O_2}} = \frac{O_1 + O_2}{20}$$

And the scaled fitness value of an individual of second type will be:

$$\frac{O_2}{20 * \frac{O_2}{O_1+O_2}} = \frac{O_1 + O_2}{20}$$

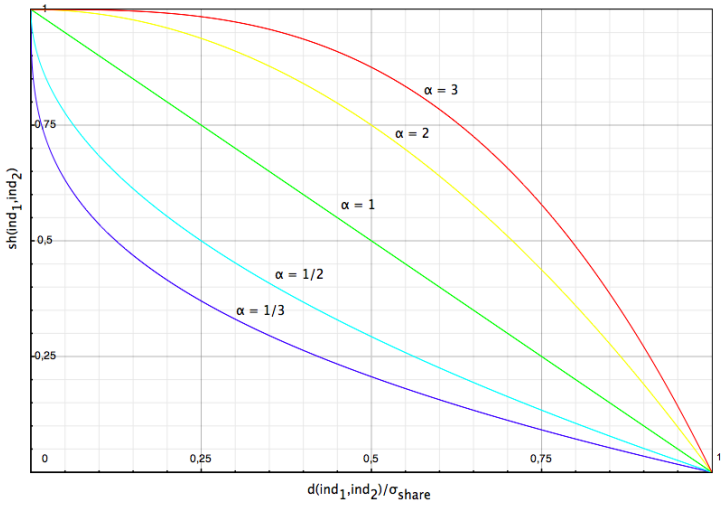
All the individuals have thus the same fitness value, and we have reached some kind of fix-point. There is no more selection pressure, and if variation operators are not too disruptive, and the population sufficiently large to reduce sampling errors and genetic drift, the ratio of individuals of type 1 and 2 will not change in the next generations.
Suppose that the search space contains n interesting solutions, the purpose of sharing is thus to prevent the population to converge to close variations of one of these solutions, and to reach an equilibrium state where the population contains n "niches" each containing close variations of one of the n solutions. The relative size of each of the niches in the population will be proportional to the fitness value of the solution they represent.

Competition will occur between variations of a same solution to occupy the niche representing this solution, so that the best local variation in the region represented by the niche will be found. Different solutions will compete to be represented by a niche in the population. Using the sharing method requires thus a sufficiently large population to put up with a potentially large number of interesting solutions to be found, and maintained against genetic drift.

The mechanism of sharing necessitates to compute the niche count of the individuals in the current generation. Typically, this niche count is computed as the sum of a sharing function value between itself and each individual in the population, including itself. This sharing function is usually calculated as:

$$sh(ind_1, ind_2) = \begin{cases} 1 - (\frac{d(ind_1, ind_2)}{\sigma_{share}})^{\alpha} & \text{if } d(ind_1, ind_2) < \sigma_{share} \\ 0 & \text{otherwise} \end{cases}$$

Where $d(ind_1, ind_2)$ refers to a distance measure between two individuals as defined in the previous subsection 1.4.2.
σ_{share} is a positive real parameter called the *niching radius*, and should correspond to the desired minimal distance between two individuals so that they can be considered as representing different solutions of the problem. If the distance value between two individuals is greater or equals to the niching radius, then the sharing function will return zero when applied to these two individuals. Otherwise, as stated above, the sharing function should return one, to count the number of similar individuals in the generation. But as the niching radius value depends on the problem, and cannot typically be established precisely, a more careful approach is adopted. The smaller the distance between the individuals compared to the niching radius is, the larger the value returned by the sharing function will be, with a maximum value of one, when the distance is zero, i.e. when the two individuals are identical. α is a positive real parameter called the scaling factor, which allows to tune how quickly the sharing function value will grow while the distance between the two individuals is reduced. Usual values for α are 1 or 2. The following graph shows the $sh(\frac{d(ind_1, ind_2)}{\sigma_{share}})$ for several values of α:



Sharing is reputed to be very efficient given a sufficiently large population size, and is one of the most often used niching technique.

The main drawback of the method is the difficulty to choose a good value for σ_{share}. If σ_{share} is larger than the distance between potentially interesting different solutions of the problem, the solutions will not be protected from competition and the algorithm probably will not find all the interesting solutions. At the opposite, if σ_{share} is too small, several niches may appear for a same solution, requiring a larger population to put up with all the interesting solutions of the problem to be found.

Note that sharing typically requires a proportional selection mechanism. As a consequence, function sh must be computed N_{gen} times per generation, where N_{gen} is the constant size of a generation, as for applying proportional selection, the total fitness of the generation must be valued. As the sh function has a $O(N_{gen})$ complexity, sharing has a time complexity of $O(N_{gen}^2)$. As sharing can require a large value of N_{gen} to be effective, the method can be very time-consuming, typically if objective and distance functions are hard to compute.

Work has been done to overcome these drawbacks. In particular, use of sharing with a tournament selection mechanism is considered in [Oei et al., 1991] and a mechanism with a better time complexity is then proposed.

[Oei et al., 1991] shows first that applying tournament selection on a current generation whose fitness has been scaled by sharing is inefficient. In theory, sharing will adapt fitness values in order to lead to a population composed of equal fitness individuals that can be grouped in niches representing each a solution of the problem. The size of a niche is proportional to the objective function value of the represented solution. In the example given above, to obtain this result, we had to consider that the fitness of each individual in the generation could be scaled perfectly to create a population of equal fitness individuals. In practice, sharing does not work so perfectly, and scaling does not modify the fitness as precisely and correctly as it has been stated above. With proportional selection, this is not too problematic, as small fitness variations do not disrupt the effect of selection that much. In tournament selection, if the best individual in the generation takes part in the tournament, it will always win. If it does not take part to the tournament, then if the second individual does take part in the tournament, it will win, and so on. Tournament selection works thus in some way by ranking the population and favoring the best ranked individuals. Small fitness variations between individuals that should have the same fitness will thus be increased by selection, and no equilibrium will be reached.

The proposed solution is *tournament selection with continuously updated sharing*. At each iteration of the generation loop, the first two individuals are selected in a normal way through deterministic tournament without any fitness scaling, and they or their offspring are added to the next generation. Next, every time a tournament is performed, the fitness of each individual that takes part in the tournament is computed by dividing the objective function value of the individual by its niche count. But this niche count is computed as if the individual was in a population composed of itself and all the individuals already copied to the next generation. Let us consider a binary tournament between an individual from niche 1 with objective value O_1 and individual from niche 2 with objective value O_2. Suppose there are k_1 individuals in the next generation members of the same niche than the first individual and k_2 for the second individual. Then the fitness value of these individuals will be respectively $\frac{O_1}{k_1+1}$ and $\frac{O_2}{k_2+1}$. The selected individual will thus be:

$$\text{Individual 1 if } \frac{O_1}{k_1+1} > \frac{O_2}{k_2+1}$$
$$\text{Individual 2 otherwise}$$

If sharing is efficient, the next generation should contain at the end a proportion of approximately $P_1 = \frac{O_1}{\sum_{niches} O_{niche}}$ individuals of niche 1, and $P_2 = \frac{O_2}{\sum_{niches} O_{niche}}$ of individuals of niche 2. Thus, we have:

$$\frac{O_1}{k_1+1} > \frac{O_2}{k_2+1} \Leftrightarrow \frac{P_1 * \sum_{niches} O_{niche}}{k_1+1} > \frac{P_2 * \sum_{niches} O_{niche}}{k_2+1} \Leftrightarrow \frac{P_1}{k_1+1} > \frac{P_2}{k_2+1}$$

If N_{gen} is the size of the population at each generation, we have finally :

$$\frac{P_1}{k_1+1} > \frac{P_2}{k_2+1} \Leftrightarrow \frac{P_1}{\frac{k_1+1}{N_{gen}}} > \frac{P_2}{\frac{k_2+1}{N_{gen}}} \Leftrightarrow \frac{\frac{k_1+1}{N_{gen}}}{P_1} < \frac{\frac{k_2+1}{N_{gen}}}{P_2}$$

And the selected individual will be:

$$\text{Individual 1 if } \frac{\frac{k_1+1}{N_{gen}}}{P_1} < \frac{\frac{k_2+1}{N_{gen}}}{P_2}$$
$$\text{Individual 2 otherwise}$$

$\frac{\frac{k_1+1}{N_{gen}}}{P_1}$ represents what ratio of the number of individuals of niche 1 sharing requires to be present in the next generation will already be present in the population if we add individual 1 to the individuals already copied to the next generation. $\frac{\frac{k_2+1}{N_{gen}}}{P_2}$ means the same for niche 2. With tournament selection with continuously updated sharing, the individual which will win the tournament will thus typically be the one whose niche has proportionally fewer individuals than it is target number (proportional to the objective function value for the niche) in the next generation. Such mechanism dynamically regulates thus the populating of the next generation as it is created in order to enforce niching effects.

To reduce the time complexity, [Oei et al., 1991] also suggests that using a sample of maximum size k of the individuals already copied to the next generation is sufficient to estimate the value of the niche count.

These ideas have been tested and exposed good results notably in [Goldberg et al., 1992].

[Sareni and Krahenbuhl, 1998] signals also that mechanisms have been developed to prevent sharing to use too disruptive variation schemes, and in that way promote stability of discovered niches, as, typically, reproduction between individuals of different niches can often produce poor quality individuals that are not members of one of the parents's niche.

1.5 Extending the framework for solving multi-objective optimization problems

1.5.1 Multi-objective optimization and Pareto optimality

With multi-objective optimization, we have no longer a single objective function o, but several objective functions o_i that all have to be maximized at the same time. The problem is different from single-objective optimization, as solutions that optimize one of the objective functions, will not necessarily optimize the other ones. Many optimization problems are multi-objective problems with conflicting objective functions. A typical example are problems where cost, performance and reliability must be optimized at the same time. In this case, a cost-wise optimal solution will typically be a less reliable and poor-performing one.
The solution to this problem is the concept of *Pareto optimality*, which refers to Vilfredo Pareto, an italian economist who invented this concept at the beginning of the twentieth century. The main idea of Pareto optimality is that a solution can only be said better than another one if it is not worse than the other one for all the objective functions and better for at least one objective function. The two solutions can also be of equivalent quality if they have the same value for all the objective functions. In the case of two solutions where one is better on at least one objective function and the other one better on at least one other objective function, the two solutions are indifferent. A formal definition of these concepts is Pareto dominance:

Definition 1.9 (Pareto dominance)**.** $\forall a, b \in S$*, the search space,*
$a \succ b$ *(a dominates b)* $\Leftrightarrow (\forall i,\ o_i(a) \geqslant o_i(b)) \wedge (\exists j,\ o_j(a) > o_j(b))$
$a = b$ *(a equals b)* $\Leftrightarrow \forall i,\ o_i(a) = o_i(b)$
$a \succeq b$ *(a weakly dominates b)* $\Leftrightarrow a \succ b \vee a = b \Leftrightarrow \forall i,\ o_i(a) \geqslant o_i(b)$
$a \sim b$ *(a is indifferent to b)* $\Leftrightarrow a \not\succeq b \wedge b \not\succeq a$

To illustrate these concepts, let us consider a search space composed of seven solutions named A, B, C, D, E, F and G. Optimization must be achieved for three objective functions: cost (to be minimized), performance (to be maximized) and reliability (to be maximized). Objective functions values for all the possible solutions are listed in the following table:

	Cost	Performance	Reliability
A	2	100	1078
B	13	154	2010
C	14	226	3456
D	9	226	3456
E	8	306	4059
F	26	227	4013
G	777	2200	54987

One can establish exhaustively which solutions dominate which other ones in the whole search space. Dominance relations in the search space are synthesized in the following table, where each row gives, for its corresponding solution, the solutions it dominates:

dominates	A	B	C	D	E	F	G
A							
B							
C							
D		X	X				
E		X	X	X		X	
F							
G							

If one looks at the columns of the previous table, they represent, for each solution, which other solutions dominate it. It can be remarked that solutions A, E and G are not dominated by any other solution in the search space. These solutions are thus the best possible solutions, as there are no solutions that are at least better for every objective function, and strictly better for at least one objective function. These non-dominated solutions are then said to be Pareto optimal, and they constitute the Pareto-optimal set.

Definition 1.10 (Pareto optimal set)**.** *Let S be the search space of a multi-objective optimization problem, and o_i with i=1...n, the n objective functions of this problem.*
The set $\{a \in S | \forall s \in S, s \not\succ a\}$ of non-dominated solutions is called the Pareto optimal set.

In multi-objective optimization, each of the n objective functions o_i can be defined as a function from S to $[0, +\infty[$ to be maximized. The space where quality can be measured for multi-objective optimization will thus be $\prod_{i=1}^{n} [0, +\infty[$. It is called the attribute space. The vectors in the attribute space that correspond to the non dominated solutions of S constitute the *Pareto front* of the problem.

Definition 1.11 (Pareto optimal front)**.** *Let S be the search space of a multi-objective optimization problem, and o_i with i=1...n, the n objective functions of this problem.*
The set $\{ \begin{pmatrix} o_1(x) \\ ... \\ o_n(x) \end{pmatrix} | x \in \{a \in S | \forall s \in S, s \not\succ a\}\}$ *is the Pareto optimal front.*

All the solutions inside the Pareto-optimal set are, as they are not dominated by any other solution in the search space, including the other Pareto-optimal solutions, either Pareto equal, or Pareto indifferent to each other. Taking two non equal Pareto optimal solutions, one will always be better than the other one on at least one objective, and vice-versa. But they can still be in some way compared. If we look at the Pareto-optimal set $\{A, E, G\}$ of our example problem, solution A is very cheap, but is very poor-performing and not reliable, solution G is high-performing and very reliable, but also very expensive, and finally solution E looks like a compromise solution. According to the context of the problem, one type of solution can eventually be preferred to another one.
Typically solving a multi-objective problem consists thus in two tasks: search and decision-making. Search means discovering the Pareto-optimal set in the search space. Decision-making consist in deciding which solution(s) in the Pareto-optimal set should be preferred. This last task cannot always be formalized a-priori, and a human decision-maker, eventually computer-assisted, is often necessary to achieve it after the search step.

1.5.2 Solving multi-objective problems with evolutionary computation

Classical techniques for solving multi-objective optimization problems consist in transforming the multi-objective problem into a single-objective one, and using a single-objective method to solve it. The transformation from multi to single objective is done by combining the different objective functions into one parameterized objective function, that will be optimized. A typically used combination is a linear combination of objectives, with parameterized linear coefficients.

Such kind of transformations can be a good way to incorporate decision making inside the search algorithm, by combining the objective functions in a way that favors some compromises among all solutions. For the linear combination scheme, one can adjust the linear coefficients in order to penalize some objectives and favor solutions that perform well on the other objectives. Such a specified preference between objectives can help reducing the search space exploration complexity. But very often, there is not enough information available on the problem and its solutions so that the way in which to combine the objective functions and to value the parameters must be established experimentally, by trial and error.

One can ask if the optima and the sufficiently good solutions of the combined objective function do really correspond to the set of Pareto-optimal solutions. Typically, the single-objective optimization algorithm is run several times with different parameters values, in order to find a set of solutions that approximate the Pareto-optimal set. For some methods, and typically the linear combination one, it can be shown that, in some cases, some solutions in the Pareto optimal set will never be found, whatever the parameters are.

As in pure single-objective optimization, for finite search space problems, exact methods or heuristics, like EAs, can be applied, as a function of the problem complexity.

Some methods also exist in order to solve multi-objective problems directly in terms of Pareto optimality, without transforming them into single-objective ones. Some researchers have suggested that evolutionary algorithms could be the best suited heuristic for such a pure multi-objective optimization [Zitzler, 1999, Page 13-14]. As EAs work on a population of solutions, they are particularly well adapted to the search of a set of solutions (the Pareto optimal one) in a single run. Moreover, the selection mechanism in evolutionary algorithms can easily be defined in term of Pareto dominance, as EAs work on a population of solutions on which domination relations can be computed. For these reasons, evolutionary computation is the most frequently used heuristic for multi-objective optimization.

When evolutionary algorithms are applied to multi-objective optimization problems, we thus have to deal with two main difficulties. First, adapted selection scheme and selection operators must be used to guide the exploration of the search space to the discovery of the Pareto-optimal set. Secondly, the population should not be allowed to converge to a single region of the search space, but to become a good sampling of the Pareto-optimal set, as well in terms of phenotypic diversity (finding different solutions), as in terms of distribution along the Pareto-front (finding diverse compromises between objectives). Mechanisms to solve these two difficulties can be incorporated in the framework defined in section 1.3, so that the encompassed GAs can solve multi-objective problems.

Many multi-objective selection mechanisms and systems to maintain diversity have been proposed and tested. We will explain in the next paragraph the *Niched Pareto Genetic*

Algorithm proposed by Horn [Horn et al., 1994], which is one of the most widely spread multi-objective GA, and the one which will be the basis for the GA developed in this thesis. A more detailed analysis of several other existing techniques can be found in the references given in the beginning of this chapter, and in [Zitzler, 1999], which has been used as a first source of information for this section. The most recent approaches for multi-objective evolutionary computation are also presented at the end of this chapter. In [Zitzler, 1999], one can also find references for reinitialization, a mechanism used in the algorithm developed in this thesis, for enforcing diversity by introducing at each generation a small number of random individuals in the population.

1.5.3 The niched Pareto genetic algorithm

The idea of using an evolutionary algorithm that combines a pure Pareto domination based selection scheme with diversity enforcing mechanisms to solve multi-objective problems in one run, was first suggested by Goldberg in the famous book "Genetic Algorithms in Search, Optimization and Machine Learning" [Goldberg, 1989].
Goldberg proposes to rank the individuals in the current generation using the Pareto domination information. The non dominated individuals within the population are given the rank #1. Then they are removed from the population, and the non dominated individuals of the remaining population are given rank #2. The process is repeated, attributing successively higher ranks until all the individuals have a rank. Traditional GA is then applied, with proportional selection operators acting on a fitness based on the rank values of the individuals. Fitness sharing, potentially coupled with mechanisms preventing disruptive variation schemes, is used to prevent premature convergence and allow to find diverse Pareto-optimal solutions.

In fact, such an approach simply adapts classical GAs with proportional selection and sharing to solve multi-objective problems. It transforms the selection mechanism to deal with Pareto dominance, while sharing is used to enforce diversity. The niched Pareto genetic algorithm adapts, in a similar way, classical GAs using tournament selection, with continuously updated sharing, to multi-objective optimization.
The adaptation proposed in the niched Pareto algorithm modifies the binary tournament selection operator with continuously updated sharing of the classical algorithm, which works now in two different steps. First, two individuals are randomly picked from the current generation to take part to the tournament, while a fixed parametric number t_{dom} of individuals are chosen in the same way, to create what Horn [Horn et al., 1994] calls a comparison set. It is then checked whether there exists individuals in this set that dominate the two candidate individuals that take part to the tournament. If one candidate is dominated and the other one is not, the non dominated individual is chosen. The second step is only applied if the two individuals are either both dominated, or both non dominated, in which case the tournament leads to a tie. The niche count of both individuals is then computed using the individuals already copied to the next generation, in the same way as in the usual tournament with continuously updated sharing mechanism. As the first part of the tournament lead to a tie, the two individuals can be considered to be of equal performance, and the individual with the lowest niche count is then the chosen one.

This mechanism allows to favor the individualsà that would be best ranked in the initial Goldberg's scheme (i.e. the less dominated ones), without having to establish the precise

ranking of each individual, which requires a lot of processing time. The size of the comparison set t_{dom} is a parameter that allows to tune the selection pressure strength: if $t_{dom} \sim N_{gen}$ then rank #1 non-dominated individuals will typically always be preferred if they face a rank > 1 dominated individual, while no distinction will be made between other rank > 1 dominated individuals. If $t_{dom} \sim 1$, then a large proportion of tournaments will lead to a tie, even when a rank #1 non-dominated individual face a rank > 1 dominated individual. The value of t_{dom} will also allow to tweak the ratio of selection pressure versus sharing pressure, as small values t_{dom} values lead to more frequent ties, where the individual with the lowest niche count is chosen, which increases sharing pressure.
As already stated, one wants to find diverse solutions that exhibit diverse compromises between the objectives. Diversity should thus be maintained in the phenotypic space, but also in the attribute space, as two very different solutions can produce similar compromise between objectives, and vice-versa. Nevertheless, the sharing method is designed considering only one space where diversity should be maintained. As Horn considers that diversity in the attribute space is the most important, the niche count is computed using a distance defined in this space. The individuals are then even more distant than their associated attribute vector, measuring how they perform for each objective, are different. [Horn et al., 1994] also proposes several untested ideas for enforcing diversity in several different spaces in the same time.
Experimental results on three test problems [Horn et al., 1994] seem to show that for $t_{dom} \approx 1\%$ of N_{gen}, too many dominated solutions stay in the population, preventing it to achieve a tight distribution of the Pareto front. With $t_{dom} \gg 20\%$ of N_{gen}, the population quickly converges to a part of the front, typically near the middle. The recommended value is $t_{dom} \approx 10\%$ of N_{gen}.

The Niched Pareto Genetic Algorithm (NPGA), is one of the oldest and of the most widely spread multi-objective implementations of GAs [Dréo et al., 2003, Page 204]. Nevertheless, many other multi-objective evolutionary implementations have been and are still developed and improved, in order to offer a better approximation of the Pareto optimal front. The most cited ones are NSGA-II [Deb et al., 2000], SPEA2 [Zitzler et al., 2001] and IBEA [Zitzler and Künzli, 2004]. These algorithms notably pay attention to use elitist mechanisms, in order to avoid loosing solutions along the search (as the Pareto-optimal set can be very large compared to the population size), but can also try to take advantage of particular multi-objective niching mechanisms especially designed for enforcing diversity in the attribute space, to allow a better integration of the decision making information into the algorithm or to reduce the algorithmic complexity.

1.6 Evaluation of evolutionary computation for optimization

1.6.1 Convergence of evolutionary algorithms

A lot of work has been done in order to analyze and model the behavior of evolutionary algorithms in a theoretical way. Rather than giving a formal and quantitative explanation of how evolutionary algorithms work, the purpose of such work is to develop predictive models able to describe a-priori how evolutionary algorithms perform on arbitrary problems. These models could thus be used in practice to tell a-priori which set of implementation

choices and parameters would be the most effective for a new given optimization problem, and what quality and speed of convergence should be expected.
By now it seems that only very limited results have been obtained. Some authors even state that no such a general encompassing explanatory and predictive theory will ever be established for evolutionary computation. The main problem faced by theoretical analysis is the huge complex nature of evolutionary algorithms mechanisms, that involve many random processes. Comparison is made with the biological fields of population genetics and evolutionary theory, which are battling against the barrier of complexity for more than a hundred years.

These elements justify the approach used in this chapter: defining evolutionary computation as a toolbox, progressively building a limited framework encompassing the tools necessary to understand and evaluate the particular algorithm developed in this thesis, and only offering a qualitative justification of the methods described, often referring to natural processes occurring in biological evolution.

If, at present, each particular evolutionary algorithm for a particular problem must and can often only be evaluated on its own limited scope, it can still be established what are the main advantages and disadvantages of EA over other optimization techniques, and on which classes of problems they have a chance to outperform other methods.

1.6.2 Advantages and disadvantages versus other optimization techniques

Evolutionary algorithms and genetic algorithms have been presented here essentially in the frame of combinatorial optimization problems, and we signaled that they could be adapted to general discrete and continuous optimization problems. We have also shown how EAs could perform well on multimodal and multi-objective optimization problems. The rich literature on evolutionary algorithms details a large and varied panel of successful applications of EAs. However it turns out that they can be very poor performing techniques on some optimization problems.
In fact, EAs are members of the class of meta-heuristics algorithms. Meta-heuristics are heuristic methods that are not especially designed for one particular optimization problem but for the whole class of optimization problems. Meta-heuristics are typically inspired by other sciences like physics (simulated annealing...), ethology (ant colonies algorithms...) and biology (evolutionary algorithm...).
[Mitchell, 1996, Section 5.1] proposes a portrait of optimization problems that could potentially benefit from meta-heuristics. First, the problem must be unfeasible to solve in an acceptable time using exact methods. On the other hand, exact methods should be preferred as they always guarantee to find optimal solutions, while heuristic methods offer no guarantee about convergence to the optimum, and can typically return a good but sub-optimal solution. If the problem is known to be unimodal, or it is known that the values of the objective functions vary smoothly as solutions are modified, hill climbing/-gradient ascent algorithms will perform better than meta-heuristics. Finally, if the search space structure is well understood, domain-specific heuristics can typically be designed and outperform general-purpose methods. Optimization problems where finding the optimal solution is not required (a sufficiently good solution can be accepted), and whose search space is large, not smooth nor unimodal, and with a structure which is not well

understood, are typical problems that are solved using meta-heuristics. The question is thus, when should EAs be preferred over other meta-heuristics ?

The No Free Lunch theorem (NFL) [Wolpert and Macready, 1997] is an often cited result that states that if we average the performance of a black-box algorithm (i.e. an algorithm using no problem specific knowledge) over all possible optimization problems, then all these kinds of algorithms will exhibit the same performance, supposing that they are programmed for not generating and testing more than once a same point of the search space. An algorithm that is high-performing on one class of problems will always pay an equivalent price with bad performances over other classes of problems. Moreover, choosing a meta-heuristic for a particular problem is by now more an art than a science, due to the lack of theoretical elements on convergence to defend a purely rational decision.
All meta-heuristics share the problem of difficult parameter tuning, and should all benefit from the integration of problem-specific knowledge. For EAs, one should pay a particular attention to develop a genotypic encoding, variation operators and a fitness function that particularly suits the underlying optimization problem.
One potential important difference of EAs compared to other search methods is that they are population algorithms, while many optimization systems are local search techniques that consider one solution at a time. This difference has already been put forward as an advantage in this chapter in the case of multimodal and multi-objective optimization.

Chapter 2

Biclustering of gene expression data

Contents

2.1 Introduction: automatic analysis of gene expression data

2.1.1 Measuring genes expression level using DNA microarrays

As explained in [NCBI, 2010], most of the biological cells that compose a living organism contain the same set of identical genes. Nevertheless, only a part of these genes are "turned on" in a cell, and it is this subset of "expressed" genes that confers particular properties to a particular cell. *Gene expression* refers to the process where the "turned on" genes in the DNA of a cell are transcribed into messenger RNA (mRNA) molecules. These molecules are subsequently translated into proteins, responsible for fulfilling the main functions of the cell. The gene expression mechanism controls which of the genes are expressed, but it can also modulate the level of expression of the expressed genes. Thanks to such a gene expression mechanism, the cell can respond in a dynamic way to environmental stimuli and to its own changing needs.
DNA microarrays are a widely used tool for analyzing gene expression in biomedical research. This technology allows to measure the level of expression of a large number of genes (and perhaps all the genes of an organism) in a sample of cells, within a single experiment. As detailed in a more extensive way in [NCBI, 2010], a microarray uses the ability of a mRNA molecule to bind specifically to any copy of the gene from which it was created. The microarray is a small and solid support onto which thousands of genes are attached in an orderly arrangement. If one measures the relative abundance of mRNA bound to every gene in the microarray, it measures in fact the level of expression of these genes in the sample from which the mRNA originates.

2.1.2 Automatic analysis of gene expression data and biclustering

The same microarray experiment, i.e. involving the same set of genes, can be applied on several samples [Berrer et al., 2003] corresponding to several relevant conditions of measurement. For example, the different samples will correspond to different points in the time evolution of the cells, or to different environmental conditions they may encounter. The cells of each sample can also come from different tissues in the organism, from different patients, etc. In that way, the expression level of a same set of genes can be measured on different experimental conditions [Madeira and Oliveira, 2004].
The result from such an experimental analysis is a large amount of data, that typically give the expression level of thousands of genes under dozens of conditions. Analyzing these expression data can allow to extract significant information from a biological (like for gene profiling) and medical (like for a better understanding of diseases) point of view [Berrer et al., 2003]. One of the usual analysis goals is to group genes that exhibit similar expression trends under some conditions [Yip, 2003]. Such a correlation between genes and conditions is a hint that these genes/conditions could be in some way related in the cellular processes. Such information can ultimately participate to the understanding of biological systems at a molecular level [Bryan, 2005, Berrer et al., 2003].

Due to the large amount of data to analyze, which could reach millions of expression values to process, the analysis is typically made using computer-assisted data mining techniques, relying on statistical and/or artificial intelligence tools [Berrer et al., 2003].

One of the first approaches used were clustering algorithms [Cheng and Church, 2000] applied either on the genes or on the conditions of the dataset. The set of expression values of a gene (respectively condition) under all the studied conditions (respectively genes) is called its gene (respectively condition) expression profile. Using a defined similarity measurement between gene expression profiles (respectively condition expression profiles), clustering algorithms partition the set of genes (respectively conditions) into a set of mutually exclusive groups or hierarchies of similar genes (respectively conditions).

The major drawback of this approach is that in order to be grouped together, a set of genes should have similar expression patterns under the whole set of conditions, and the other way round. However our understanding of the cellular processes [Madeira and Oliveira, 2004] leads us to expect that a subset of genes can have similar expression patterns only under some conditions, and behave independently for the other conditions, and the other way round. For example [Yip, 2003], a same set of genes can have a similar response to a given environmental stimulus, but each gene can have some different functions at other times. Similarly, for a set of related conditions, like a set of samples of tumorous tissues, some genes may exhibit different expression patterns, for example if the tumors are of different sub-types.

To address this limitation, Cheng and Church [Cheng and Church, 2000] proposed a formal specification and an algorithmic method to perform what they called biclustering, i.e. clustering of both genes and conditions simultaneously. Informally, the goal of biclustering is thus to search the data for groups of genes and conditions where the genes of the group exhibit highly correlated expression levels only for every condition of the group. Such a group is called a *bicluster*.

Many different formal specifications, taking into account many biological and computational aspects of the problem, and many numerical techniques to find biclusters in gene expression data have been developed since then, and are still developed now. Surveys of some of the existing specifications and techniques can be found notably in [Yip, 2003] and [Madeira and Oliveira, 2004]. Using the formal specification of the biclustering problem proposed by Cheng and Church, F. Divina [Divina and Aguilar-Ruiz, 2006] [Divina and Aguilar-Ruiz, 2007] developed such a biclustering technique using genetic algorithms. In this thesis we use existent evolutionary computation techniques and propose and test a new technique to improve this GA-based biclustering technique.

In this chapter, we first detail (section 2.2) the δ-bicluster model, proposed by Cheng and Church, which defines a formal and mathematical specification of the biological biclustering problem. We also say a few words on the biclustering algorithm proposed by Cheng and Church, and on its experimental testing. In the context of the δ-bicluster model, we present then the SEBI and the SMOB biclustering genetic algorithms proposed by F. Divina (section 2.3).

Before giving further details on the formal specification of biclustering proposed by Cheng and Church, we end this introduction by reviewing first the methodological validity of the biclustering approach. Secondly, we list some of the other contexts, different from expression data analysis, where automatic biclustering techniques, as the one we developed in this thesis, could be useful.

2.1.3 Methodological validity of the biclustering approach

The scientific methodology that underlies the biclustering of gene expression data is very simple. First, a gene expression dataset is collected using the microarray technique. Then a mathematical and formal model defining what a bicluster is, i.e. specifying what kind of correlation between data should be searched for in the dataset, is proposed. The dataset is then analyzed by automatic biclustering techniques, to find biclusters as defined by the proposed model. The discovered biclusters are then used as a hint of some relation at the cellular level between some genes and conditions.
At least four main validity questions can be asked about this methodology: the validity of the microarray experiments, the quality of the results returned by these experiments, the relevance of the bicluster model and the significance of the biclusters found.

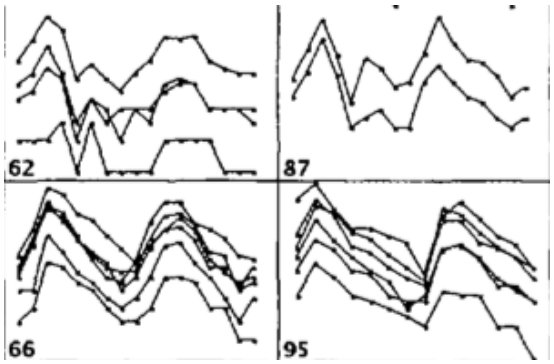
The first question to be raised is the validity of the microarray experiments, and of their underlying theoretical biological background. Such a question is out of the scope of this work. The effectiveness of the microarrays technique seems nevertheless widely admitted in the biological field [Berrer et al., 2003].

The quality of the obtained expression levels data should also be discussed. Two main issues should be considered here [Berrer et al., 2003]: *missing data*, and *measurement variations*.
The measurement of the expression level of a particular gene for a particular condition can fail for many reasons. In this case, we must deal with the problem of missing data. From a pure data analysis point of view, two solutions for this problem are possible. Either the missing datum is ignored, or it is replaced by a reasonable or plausible value.
Measurement variations occur for two typical reasons. First, imperfections in the instruments, processes and materials involved in microarray experiments will typically cause errors in the measurement of expression levels. Moreover, natural variations in the studied biological processes make measurement not entirely deterministic. The main approach to deal with such noise in the data is replication of the experiment.
However, each condition in the data correspond to a particular microarray experiment, where the expression level of the set of genes is measured on a particular sample. The data is thus composed by combining the data of many distinct microarray experiments. Inevitable measurement variations between these different experiments should be taken into account, in order to place each experiment on a comparable scale. Some numerical procedures known as global *normalization* can be designed to solve this problem. A more throughout discussion of this topic can be found in [Berrer et al., 2003].

The relevance of the chosen bicluster model is a complex question. It should be first verified the validity of the concept of biclustering from a biological point of view. Then, the adequacy of the translation from this biological concept to a formal bicluster model should be established. Finally, the ability of the model to be implemented into an effective computer program should be taken into account.
In this work, we will use the Cheng and Church's δ-bicluster model as a reference, which is considered both by biologists [Berrer et al., 2003] and computer scientists [Madeira and Oliveira, 2004].

Finally, once a bicluster has been discovered, it should be evaluated whether the correlation between the expression levels of some genes under some conditions established by the

bicluster is statistically significant. If it is, it should be verified whether this correlation is really a witness of some underlying biological process or not.
Statistical significance is typically verified by defining some statistical correlation score and a corresponding significance threshold score. Adequate visualization techniques are also widely used to reveal correlation patterns. For example, the following figure is extracted from [Cheng and Church, 2000]. Each of the four bicluster expression graphs corresponds to a discovered bicluster in an experimental dataset. For each gene in the bicluster, the expression level (y-axis) of the gene is reported for each condition (x-axis) in the bicluster. Each discrete curve in the graph links the points corresponding to a same gene of the bicluster. The visual inspection of these graphs reveals an apparent correlation between the evolutions of the expression level of the genes for the different conditions.



Four sample bicluster expression graphs

A discussion of the biological interpretation of the statistically significant discovered biclusters is out of the scope of this work.

2.1.4 Other applications of automatic biclustering techniques

The automatic biclustering methods, as the one developed in this thesis, are basically data mining techniques, and could be useful in many applications other than the analysis of gene expression data. [Madeira and Oliveira, 2004] lists some of the current applications of biclustering techniques:

Recommendation systems and target marketing Biclustering techniques have be applied to marketing data, in order to find subgroups of customers with similar preferences or behaviors [Yang et al., 2002, Wang et al., 2002] [Hofmann and Puzicha, 1999, Ungar and Foster, 1998].

Information retrieval and text mining Biclustering methods are used to identify subgroups of documents with similar properties for some groups of attributes, like words or images, which can be a task of particular importance for the development of search engines [Dhillon, 2001, Berkhin and Becher, 2002].

Dimensionality reduction in large databases Authors have taken advantage of biclustering algorithms to find sub-groups of rows in a database table, that exhibit similar trends for some of the columns of the table [Agrawal et al., 1998].

Analysis of electoral data Other researchers have applied automatic biclustering systems on electoral data to find subgroups of citizens with the same political ideas and electoral behaviors [Hartigan, 1972].

Others Many other data in many different domains could potentially benefit from biclustering techniques. For example, [Lazzeroni and Owen, 2000] used biclustering techniques to analyze nutritional and foreign exchange data.

These examples suggest that biclustering techniques can exhibit the same limitations and risks as many other data mining techniques. Such techniques typically try to induce some general behavior from a large number of individual examples. The veracity of the induction should typically be put into question, as many interpretation errors can be made. For example, some hidden factors could have been forgotten, or the induction made could be influenced by some a-priori intuition or intention.

2.2 The δ-bicluster model and the Cheng and Church's algorithm

In [Cheng and Church, 2000], Cheng and Church propose an algorithmic method for biclustering gene expression data. For such an algorithmic solution to be possible, they present first a formal and mathematical specification of this problem, known as the δ-bicluster model.
In this section we detail first this δ-bicluster model, using notably the details presented in [Madeira and Oliveira, 2004] and [Yang et al., 2003]. For this purpose, we start with an informal definition of the biclustering problem, which summarizes the details exposed in the introduction of this chapter.

Definition 2.1 (Biclustering problem (Informal definition))**.** *Given a set of data measuring the expression level of a given set of genes under a given set of conditions, find groups of genes and conditions, where, for each group, all the genes of the group exhibit a similar expression trend under all the conditions of the group.*

In the next paragraphs, we use a stepwise approach where each step refines and formalizes the previous definition. Step by step, we build in that way the formal specification of biclustering proposed by Cheng and Church.

2.2.1 The expression matrix

Gene expression data can typically be organized in an *expression matrix* EM. Each row of this matrix corresponds to a gene studied in the experimental analysis. Similarly, each column of the matrix corresponds to a studied condition. We will suppose that the problems of missing data and of measurement variations have been solved for the considered data. Each element EM_{ij} of the matrix is then a real number that measures the expression level of gene i under condition j. The order in which the genes and the conditions are situated in the matrix has no particular meaning. From now on, we will suppose that this order has been arbitrarily fixed and we will refer to genes and conditions using the number of their corresponding row/column.
Let us call respectively G and C the set of the genes and conditions studied in the experimental analysis. At each group of genes $S \subseteq G$ and group of conditions $M \subseteq C$ corresponds a sub-matrix (S, M) of the expression matrix EM, composed only of the rows and columns of EM corresponding respectively to the genes in S and the columns in M.
The following figure represents a sample expression matrix EM (on the left). Each row represents a numbered gene, and each column a numbered condition. An element in the

matrix corresponds to the expression level of the gene of its row under the condition of its column. An example of a sub-matrix (S, M) of EM is visible on the right of the figure. For this particular sub-matrix, $S = \{G4, G8, G15, G32, G34\}$ and $M = \{C3, C4, C7, C9\}$. (S, M) is thus composed of the green elements of EM, at the intersection of the yellow rows corresponding to genes in S and of the blue columns corresponding to conditions in M.

EM

	C1	C2	C3	C4	C5	C6	C7	C8	C9
G1	88,6	19,71	20,78	25,54	48,63	35,93	17,88	37,76	74,49
G2	74,17	10,67	41,03	30,85	63,42	7,652	18,98	5,231	9,893
G3	86,03	78,92	1,873	63,92	64,38	92,64	31,57	77,93	63,65
G4	39,65	24,92	16,25	92,92	69,59	54,29	42,71	29,79	69,69
G5	21,71	96,47	33,69	14,26	93,6	80,49	62,91	71,91	57,95
G6	37,2	63,32	9,844	4,488	83,03	39,53	80,25	79,08	52,93
G7	91,72	45,91	15	84,85	51,6	38,48	29,43	13,53	75,62
G8	44,69	13,34	1,747	50,49	66,4	70,05	91,21	34,15	45,73
G9	16,49	17,29	72,77	48,79	14,8	19,31	44,24	5,262	70,15
G10	26,16	66,17	30,8	97,7	36,41	83,47	8,905	42,87	17,64
G11	56,24	68,59	91,33	78,52	69,35	78,08	83,39	54,6	21,32
G12	36,73	15,91	80,96	10,95	62,13	92,4	44,17	12,27	41,77
G13	52,41	14,13	50,36	90,31	90,95	0,503	63,65	91,17	62,17
G14	98,65	64,92	79,81	4,075	66,79	21	57	82,47	89,8
G15	61,92	86,72	60,45	94,13	18,09	56,55	40,66	70,52	73,73
G16	69,46	43,05	40,36	34,36	52,78	24,68	71,81	88,15	31,01
G17	53,03	89,46	20,17	52,42	34,84	31,82	83,47	10,87	94,19
G18	72,87	48,55	50,71	9,508	13,93	19,73	48,02	51,71	70,38
G19	42,07	31,61	71,75	62,53	27,65	9,228	0,336	62,74	79,94
G20	45,78	0,788	64,7	31,79	69,94	66,14	83,7	49,49	15,61
G21	99,15	56,63	40,6	66,88	25,71	22,54	59,85	86,85	23,98
G22	43,69	49,85	8,739	91,54	58,53	86	82,12	30,62	25,34
G23	98,48	23,95	30,48	3,435	6,907	6,869	67,77	76,01	2,307
G24	82,24	12,81	26,93	48,68	62,31	66,36	48,44	99,39	16,49
G25	82,22	18,25	54,99	53,46	53,93	16,68	6,909	91,66	70,93
G26	59,06	35,57	85,88	5,16	61,57	62,33	65,27	9,17	20,47
G27	31,9	85,39	88,92	55,91	4,35	91,27	67,94	13,18	51,85
G28	60,43	16,35	28,26	78,35	48,46	32,64	65,34	69,85	20,56
G29	36,8	67,75	67,32	20,06	44,98	95,24	24,94	68,68	89,3
G30	58,91	89,42	96,38	34,08	16,51	40,95	29,12	53,1	76,56
G31	32,7	68,32	29,04	97,2	85,52	69,86	62,77	29,82	17,54
G32	3,214	62,67	80,53	4,447	92,21	27,21	35,54	31,05	65,46
G33	61,89	82,04	86,68	29,06	83,04	91,74	83,04	1,353	30,74
G34	11,56	25,88	21,21	34,32	52,13	2,42	53,99	94,66	90,07

(S,M)

	C3	C4	C7	C9
G4	16,25	92,92	42,71	69,69
G8	1,747	50,49	91,21	45,73
G15	60,45	94,13	40,66	73,73
G32	80,53	4,447	35,54	65,46
G34	21,21	34,32	53,99	90,07

Biclustering the expression data means finding biclusters in it. A bicluster has been defined as a group of genes that exhibit similar expression patterns in the data under a given set of conditions. Using the formalization introduced, a bicluster to be found is a group of at least two genes $S \subseteq G$ and a group of at least two conditions $M \subseteq C$ so that the elements of the corresponding sub-matrix (S, M) exhibit some coherent tendency. This allows us to propose a first formal refinement of the definition the biclustering problem.

Definition 2.2 (Biclustering problem (Refinement step #1))**.** *Given an expression matrix EM organizing gene expression data, find sub-matrices (S, M) of EM, with at least two rows and two columns, whose elements exhibit some coherent tendency.*

We can now remark that in order to solve algorithmically the biclustering problem, we should be able to evaluate in a formal way the coherence of any sub-matrix of (E, M). Such a formal measure is proposed by Cheng and Church as a coherence score, called the *mean squared residue*, which we establish in the following subsection.

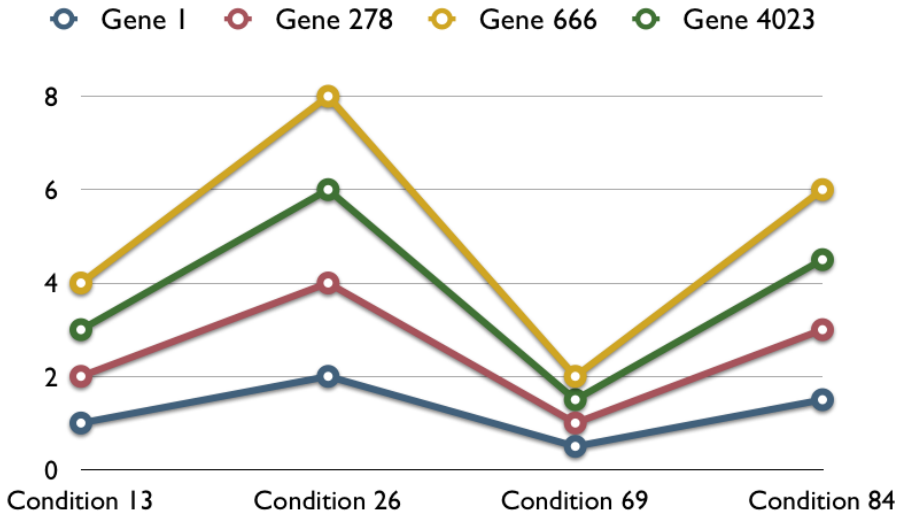
2.2.2 Measuring the coherence of a sub-matrix

A Criterion of coherence

The mean squared residue of a sub-matrix basically measures how incoherent a sub-matrix is, compared to the definition of a perfectly coherent coherent sub-matrix. Cheng and Church consider a sub-matrix to be perfectly coherent if each row of the sub-matrix (i.e. each considered gene expression pattern under the set of considered conditions) can be obtained by multiplying each of the other by a constant value. Biologically, this criterion means that, for each couple of conditions in the sub-matrix, the ratio between the abundance of mRNA produced by a gene of the sub-matrix for the first condition and the abundance of mRNA produced by the same gene for the second condition will be constant for all genes in the matrix.
The following figure shows an example of such a perfectly coherent sub-matrix. One can remark that row 1 equals $\frac{1}{2} *$ row 2, $\frac{1}{4} *$ row 3 and $\frac{1}{3} *$ row 4. As a consequence, each column can also be obtained by multiplying another one by a constant value: column 1 equals $\frac{1}{2} *$ column 2, $2 *$ column 3 and $\frac{2}{3} *$ column 4.

	Condition 13	Condition 26	Condition 69	Condition 84
Gene 1	1.0	2.0	0.5	1.5
Gene 278	2.0	4.0	1.0	3.0
Gene 666	4.0	8.0	2.0	6.0
Gene 4023	3.0	6.0	1.5	4.5

The following graph is a bicluster expression graph similar to the one described in subsection 2.1.3. It offers a visualization of the data of the perfectly coherent sub-matrix defined in the previous figure. For each gene of the sub-matrix, it shows the discrete curve of its expression level under each of the conditions of the sub-matrix. Visual inspection reveals obviously the coherence of the sub-matrix.



Mathematically, one can easily see that a perfectly coherent sub-matrix (S, M) can be described as a sub-matrix in which each element $(S, M)_{ij}$ equals the product between a

typical value within the sub-matrix denoted by $\mu_{(S,M)}$ and an adjustment value for its row i, α_i, and an adjustment value for its column j, β_j :

$$(S,M)_{ij} = \mu_{(S,M)} * \alpha_i * \beta_j \tag{2.1}$$

In the previous example, values for $\mu_{(S,M)}$, α_i and β_j could be $\mu_{(S,M)} = 1$, $\alpha_1 = 1$, $\alpha_2 = 2$, $\alpha_3 = 4$, $\alpha_4 = 3$, $\beta_1 = 1$, $\beta_2 = 2$, $\beta_3 = 0.5$ and $\beta_4 = 1.5$.

B The log-transformed expression matrix

Cheng an Church propose to log-transform the values in the expression matrix. They work thus with an expression matrix EM' that contains the logarithm of the expression level of the genes instead of this expression level itself, i.e. $EM'_{ij} = log(EM_{ij})$. This practice is widely used by biologists as a part of the normalization of microarray experiments data [Berrer et al., 2003, Page 79].
Using equation 2.1 and the classical properties of logarithms, we can establish what becomes the value of an element of a perfectly coherent sub-matrix, when the EM matrix has been log-transformed:

$$(S,M)'_{ij} = log((S,M)_{ij}) = log(\mu_{(S,M)} * \alpha_i * \beta_j) = log(\mu_{(S,M)}) + log(\alpha_i) + log(\beta_j)$$

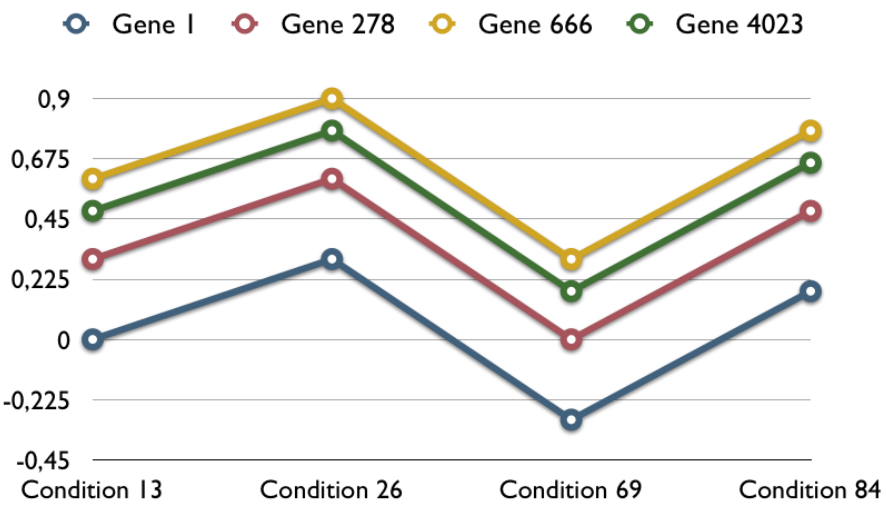
In the log-transformed matrix EM', a perfectly coherent sub-matrix is thus a sub-matrix in which each element $(S,M)'_{ij}$ equals the sum between a typical value within the sub-matrix $\mu'_{(S,M)'} = log(\mu_{(S,M)})$ and an adjustment value for its row i, $\alpha'_i = log(\alpha_i)$, and an adjustment value for its column j, $\beta'_j = log(\beta_j)$:

$$(S,M)'_{ij} = \mu'_{(S,M)'} + \alpha'_i + \beta'_j$$

This means that each row and thus each column of the sub-matrix can be obtained by adding a constant value to each of the other. This can be illustrated with the log-transformed (base 10 has been used) version of the sample sub-matrix defined above (log values have been rounded):

	Condition 13	Condition 26	Condition 69	Condition 84
Gene 1	0.0	0.30	-0.30	0.18
Gene 278	0.30	0.60	0.0	0.48
Gene 666	0.60	0.90	0.30	0.78
Gene 4023	0.48	0.78	0.18	0.66

One can remark that row 1 equals $log(\frac{1}{2})$ + row 2, $log(\frac{1}{4})$ + row 3 and $log(\frac{1}{3})$ + row 4. Similarly column 1 equals $log(\frac{1}{2})$ + column 2, $log(2)$ + column 3 and $log(\frac{2}{3})$ + column 4. The log-transformation transforms thus multiplicative changes into additive increments. This appears obviously for the rows by visualizing the bicluster expression graph of the log-transformed sub-matrix, where the rows appear shifted by constant offsets:



C Correlation between elements in a perfectly coherent sub-matrix

In a log-transformed expression matrix, it is easy to see that one can also define a perfectly coherent sub-matrix, as a sub-matrix where any element $(S, M)'_{ij}$ can be calculated from any other element $(S, M)'_{kl}$ by adding $(S, M)'_{kl}$ the offset between row i and row k, δ_{ik}, and the offset between column j and column l, δ^{jl}:

$$(S, M)'_{ij} = (S, M)'_{kl} + \delta_{ik} + \delta^{jl} \tag{2.2}$$

For example, in the sample log-transformed sub-matrix defined above, element $(S, M)'_{32} = 0.90$ equals element $(S, M)'_{11} = 0.0$ plus the offset between row 3 and row 1, $\delta_{31} = -log(\frac{1}{4}) = 0.60$, and plus the offset between column 2 and column 1, $\delta^{21} = -log(\frac{1}{2}) = 0.30$.

Using equation 2.2 summing k over all the s rows of $(S, M)'$ and summing l over all the m columns of $(S, M)'$, we obtain:

$$(s * m) * (S, M)'_{ij} = \sum_{k=1}^{s} \sum_{l=1}^{m} ((S, M)'_{kl} + \delta_{ik} + \delta^{jl})$$

Which is equivalent to:

$$(S, M)'_{ij} = (\frac{1}{s * m}) * (\sum_{k=1}^{s} \sum_{l=1}^{m} (S, M)'_{kl} + m * \sum_{k=1}^{s} \delta_{ik} + s * \sum_{l=1}^{m} \delta^{jl}) \tag{2.3}$$

If we call $\mathcal{M}_i$ the mean of the elements of row i of the sub-matrix, and $\mathcal{M}_k$ the mean of the elements of row k of the sub-matrix, we have:

$$\begin{aligned}
\mathcal{M}_i - \mathcal{M}_k &= (\frac{1}{m} * \sum_{o=1}^{m}(S,M)'_{io}) - (\frac{1}{m} * \sum_{o=1}^{m}(S,M)'_{ko}) \\
&= (\frac{1}{m}) * \sum_{o=1}^{m}((S,M)'_{io} - (S,M)'_{ko}) \\
&= (\frac{1}{m}) * \sum_{o=1}^{m}\delta_{ik} \\
&= (\frac{1}{m}) * (m * \delta_{ik}) \\
&= \delta_{ik} \qquad (2.4)
\end{aligned}$$

Similarly, if $\mathcal{M}^j$ is the mean of the elements of column j of the sub-matrix, and $\mathcal{M}^l$ the mean of the elements of column l of the sub-matrix, we have:

$$\mathcal{M}^j - \mathcal{M}^l = \delta^{jl} \qquad (2.5)$$

Finally, if $\mathcal{M}$ is the mean of all the elements in the sub-matrix $(S,M)'$, we have:

$$\mathcal{M} = (\frac{1}{s*m}) * \sum_{k=1}^{s}\sum_{l=1}^{m}(S,M)'_{kl} \qquad (2.6)$$

Injecting equations 2.4, 2.5 and 2.6 in 2.3, we obtain:

$$\begin{aligned}
(S,M)'_{ij} &= \mathcal{M} + \frac{1}{s} * \sum_{k=1}^{s}(\mathcal{M}_i - \mathcal{M}_k) + \frac{1}{m} * \sum_{l=1}^{m}(\mathcal{M}^j - \mathcal{M}^l) \\
&= \mathcal{M} + \frac{1}{s} * ((s * \mathcal{M}_i) - \sum_{k=1}^{s}\mathcal{M}_k) + \frac{1}{m} * ((m * \mathcal{M}^j) - \sum_{l=1}^{m}\mathcal{M}^l)) \\
&= \mathcal{M}_i + \mathcal{M}^j + \mathcal{M} - \frac{1}{s} * \sum_{k=1}^{s}\mathcal{M}_k - \frac{1}{m} * \sum_{l=1}^{m}\mathcal{M}^l \qquad (2.7)
\end{aligned}$$

As $\mathcal{M}$ is the mean of the whole sub-matrix, $\mathcal{M}_k$ is the mean of row k, and $\mathcal{M}^l$ the mean of column l, we have, by definition:

$$\mathcal{M} = \frac{1}{s} * \sum_{k=1}^{s}\mathcal{M}_k = \frac{1}{m} * \sum_{l=1}^{m}\mathcal{M}^l \qquad (2.8)$$

Injecting equation 2.8 in 2.7, we obtain:

$$\begin{aligned}
(S,M)'_{ij} &= \mathcal{M}_i + \mathcal{M}^j + \mathcal{M} - \mathcal{M} - \mathcal{M} \\
&= \mathcal{M}_i + \mathcal{M}^j - \mathcal{M} \qquad (2.9)
\end{aligned}$$

Equation 2.9 is simply a rewriting of the definition of a perfectly coherent sub-matrix in a log-transformed expression matrix, given at equation 2.2. We can thus define such a perfectly coherent sub-matrix, using equation 2.9, as a sub-matrix whose each elements equal the sum of the mean of its row and of its column, minus the mean of the whole sub-matrix.

D The mean squared residue

The purpose of this section was to define the measure of coherence of a sub-matrix proposed by Cheng an Church, the mean squared residue. The mean squared residue of a sub-matrix measures how incoherent the sub-matrix is, compared to the definition of a perfectly coherent coherent sub-matrix. Now that we have formally defined what a perfectly coherent sub-matrix is, we can measure how much a given sub-matrix is incoherent, compared to this definition.

Let us work by means of example with a very small 2×2 sub-matrix $(S, M)'$, which is not perfectly coherent:

$$(S, M)' = \begin{array}{|c|c|}\hline 5 & 2 \\ \hline 1 & 3 \\ \hline \end{array}$$

We can compute the row means, column means, and the global mean of this sub-matrix: $\mathcal{M}_1 = 3.5$, $\mathcal{M}_2 = 2$, $\mathcal{M}^1 = 3$, $\mathcal{M}^2 = 2.5$, $\mathcal{M} = 2.75$. Using equation 2.9, we can create the only 2×2 sub-matrix that has the same row means and column means as $(S, M)'$, and which is perfectly coherent:

$$\begin{array}{rcl} (S, M)'_{coherent} & = & \begin{array}{|c|c|}\hline \mathcal{M}_1 + \mathcal{M}^1 - \mathcal{M} & \mathcal{M}_1 + \mathcal{M}^2 - \mathcal{M} \\ \hline \mathcal{M}_2 + \mathcal{M}^1 - \mathcal{M} & \mathcal{M}_2 + \mathcal{M}^2 - \mathcal{M} \\ \hline \end{array} \\ & = & \begin{array}{|c|c|}\hline 3.75 & 3.25 \\ \hline 2.25 & 1.75 \\ \hline \end{array} \end{array}$$

Comparing $(S, M)'$ and $(S, M)'_{coherent}$ allows to measure how much the values of the elements in $(S, M)'$ must be changed, to render $(S, M)'$ perfectly coherent, without affecting the row means and column means. The difference between $(S, M)'$ and $(S, M)'_{coherent}$ seems thus to be a relevant measure of the coherence of $(S, M)'$.
Cheng and Church define the residue res of an element $(S, M)'_{ij}$ of $(S, M)'$, as the difference between $(S, M)'_{ij}$ and the corresponding element in $(S, M)'_{coherent}$, $((S, M)'_{coherent})_{ij}$. $((S, M)'_{coherent})_{ij}$ can be computed from the row mean and column mean of $(S, M)'_{ij}$ and from the global mean of $(S, M)'$:

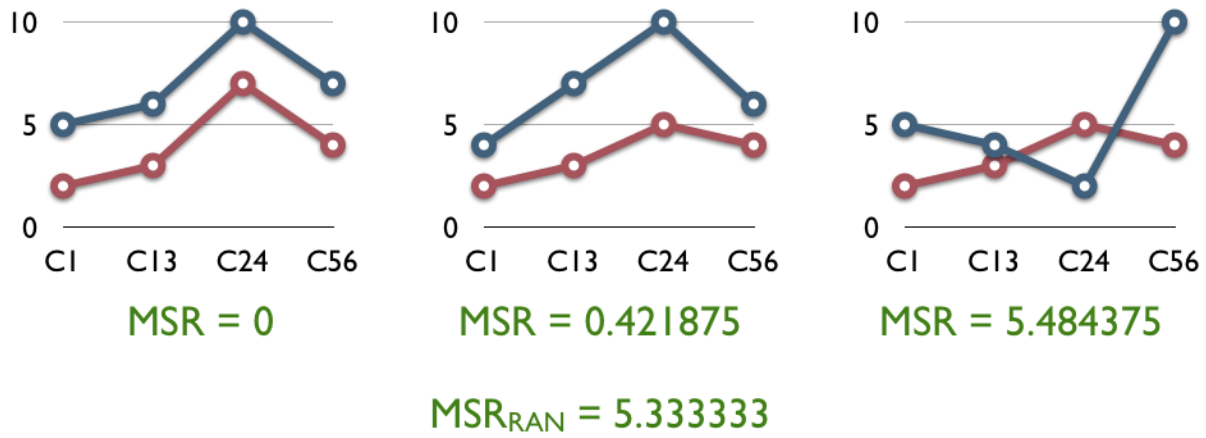
$$\begin{array}{rcl} res((S, M)'_{ij}) & = & (S, M)'_{ij} - ((S, M)'_{coherent})_{ij} \\ & = & (S, M)'_{ij} - \mathcal{M}_i - \mathcal{M}^j + \mathcal{M} \end{array}$$

Cheng and Church then measure the coherence of a sub-matrix using the mean squared residue, i.e. the mean of the squared residue of all the elements in the sub-matrix. The mean squared residue is indeed an effective measure of the difference between $(S, M)'$ and $(S, M)'_{coherent}$.

Definition 2.3. *The **mean squared residue** (MSR) is a positive real-valued function that associates each sub-matrix of a log-transformed expression matrix EM' a score measuring the inverse of its coherence, defined as:*

$$MSR((S, M)') = \frac{1}{s * m} \sum_{i=1}^{s} \sum_{j=1}^{m} (res((S, M)'_{ij}))^2$$

The MSR of a perfectly coherent sub-matrix is obviously zero. The more a sub-matrix is incoherent, the more its MSR score will be high. Cheng and Church note also that a sub-matrix composed of elements randomly and uniformly generated in the range [a,b] has an expected score MSR_{ran} of $\frac{(b-a)^2}{12}$, independently of the size of the sub-matrix.
The concept of MSR score is illustrated on the figure below. Graphs representing three different sub-matrices are shown with their corresponding MSR. The left graph corresponds to a perfectly coherent sub-matrix. The middle graph corresponds to a relatively coherent sub-matrix. The right graph corresponds to a completely incoherent sub-matrix. MSR_{ran} which is similar for the three sub-matrices is also indicated.



E δ**-biclusters**

Having defined a measure of the coherence of a sub-matrix, the MSR, Cheng and Church propose to introduce a MSR threshold δ under which a sub-matrix is considered as significantly coherent. Such a sub-matrix is then called a *δ-bicluster*. Taking this formalization into account, we can refine the specification of the biclustering problem given at definition 2.2:

Definition 2.4 (Biclustering problem (Refinement step #2))**.** *Given a log-transformed expression matrix EM' organizing gene expression data, find sub-matrices $(S, M)'$ of EM', with at least two rows and two columns, and whose mean squared residue (MSR) is lower that a given threshold δ.*

2.2.3 Size of the δ-biclusters

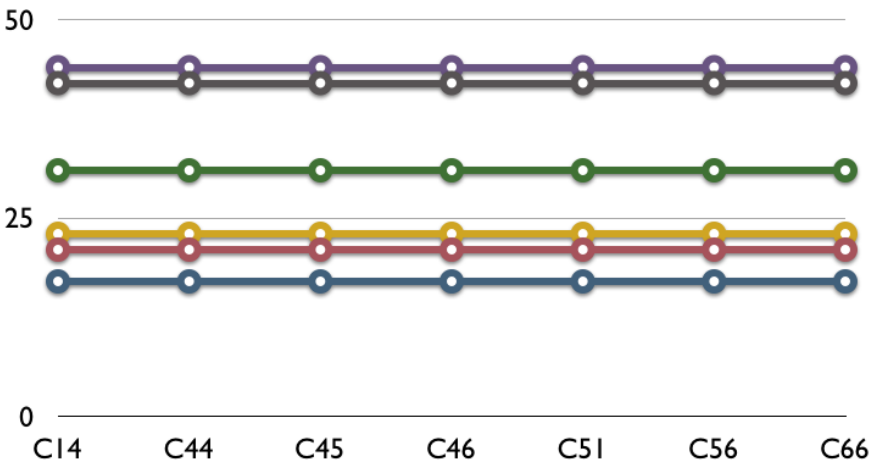
A δ-bicluster is thus a sub-matrix composed of a set of sufficiently correlated rows and a set of sufficiently correlated columns. Once a δ-bicluster has been found, if one removes a row or a column from this δ-bicluster, the resulting sub-matrix will obviously still be composed of rows and columns that are still correlated with each other. This resulting sub-matrix has thus chance to be also a δ-bicluster. If a biclustering algorithm returns this new δ-bicluster instead of the first one, it would lose an important piece of information: the fact that the removed row/column is indeed correlated with the other ones in the bicluster.
We can define the size of a sub-matrix as the product of its number of rows and its number of columns. An effective biclustering means thus to find δ-biclusters of maximal size. We should then refine again our specification of the biclustering problem.

Definition 2.5 (Biclustering problem (Refinement step #3))**.** *Given a log-transformed expression matrix EM' organizing gene expression data, find sub-matrices $(S, M)'$ of EM' of maximal size, with at least two rows and two columns, and whose mean squared residue (MSR) is lower that a given threshold δ.*

Nevertheless, Cheng and Church show that when a sub-matrix has a non-zero MSR, it is always possible to remove a row or a column to lower this MSR. Size and MSR seem thus to be antagonist criteria. If we choose a low value for the MSR threshold δ, it will allow us to find very coherent δ-biclusters, but it may be at the expense of the maximality of the discovered δ-biclusters. On the contrary, we can choose a higher MSR threshold δ, in order to find larger δ-biclusters, but it may be at the expense of the coherence and thus of the biological significance of the discovered δ-biclusters.

2.2.4 Avoiding flat δ-biclusters

One solution of the biclustering problem as specified at definition 2.5 is a flat or nearly flat sub-matrix, i.e. a sub-matrix where the expression level for each gene is constant or nearly constant for all the conditions. A graph illustrating such a sub-matrix is available in the following figure. One can see that each row (respectively column) of the sub-matrix can obviously be computed by adding a constant value to each of the the other rows (respectively columns). A flat sub-matrix fits thus the definition of a perfectly coherent sub-matrix we detailed previously. As a consequence, such a sub-matrix will have a zero MSR and will be considered as very good δ-bicluster by the specification.



In fact, these kinds of δ-biclusters are not very interesting from a biological point of view. [Cheng and Church, 2000] notes that "In expression data analysis [...] more interesting is the finding of a set of genes showing strikingly similar up-regulation and down-regulation under a set of conditions".
The variance is statistical measure of the dispersion of a set of values around their mean. The more the values in the set are close from each other, the more the variance is low, and the other way round. As a set of equal values has a zero variance, the variance of the elements in a row of flat δ-bicluster will be zero. The more the genes will exhibit strongly varying expression levels under the set of conditions in the sub-matrix, the more the variance of the elements in their row ("row variance") will be high. A score to measure the flatness of a δ-bicluster is thus the mean of the row variance for each of the row in the sub matrix. This score is called the mean row variance:

Definition 2.6. *The **mean row variance** (MRV) is a positive real-valued function that associates each sub-matrix of a log-transformed expression matrix EM' a score measuring its flatness, defined as:*

$$MRV((S,M)') = \frac{1}{s*m}\sum_{i=1}^{s}\sum_{j=1}^{m}((S,M)'_{ij} - \mathcal{M}_i)^2$$

A sub-matrix with both a low MSR and a strong MRV has a lot of chances to show "strikingly similar up-regulation and down-regulation [of the genes] under [the] set of conditions", as described by Cheng and Church. We can thus refine once again the specification and finally propose the final formal specification of the biclustering problem, as defined by Cheng and Church.

Definition 2.7 (Biclustering problem (Cheng and Church's formal specification))**.** *Given a log-transformed expression matrix EM' organizing gene expression data, find sub-matrices $(S,M)'$ of EM' of maximal size and sufficiently high mean row variance (MRV), with at least two rows and two columns, and whose mean squared residue (MSR) is lower that a given threshold δ.*

In this definition, we have:

$$MSR((S,M)') = \frac{1}{s*m}\sum_{i=1}^{s}\sum_{j=1}^{m}((S,M)'_{ij} - \mathcal{M}_i - \mathcal{M}^j + \mathcal{M})^2$$

$$MRV((S,M)') = \frac{1}{s*m}\sum_{i=1}^{s}\sum_{j=1}^{m}((S,M)'_{ij} - \mathcal{M}_i)^2$$

$$size((S,M)') = s*m$$

with

- s the number of rows in the sub-matrix $(S,M)'$.
- m the number of columns in the sub-matrix $(S,M)'$.
- $\mathcal{M}_i$ the mean of the elements of row #i of $(S,M)'$.
- $\mathcal{M}^j$ the mean of the elements of column #j of $(S,M)'$.
- $\mathcal{M}$ the mean of all the elements of $(S,M)'$.

2.2.5 Algorithmic complexity

Now that we have established the formal specification of the biclustering problem proposed by Cheng and Church, we can ask the question of its algorithmic complexity. For this paragraph, we will temporarily leave the context of biology, and consider biclustering from a pure computer science point of view. For a computer scientist, biclustering gene expression data is only a particular instance of a more general optimization problem: finding a set of sub-matrices of a given data matrix, such that each sub-matrix optimizes a set of quality criteria, enforcing essentially a coherence requirement [Madeira and Oliveira, 2004].
The algorithmic complexity of the instance of this general problem we defined at definition 2.7 has not been formally established. Nevertheless, most instances of the general

problem has been proved to be NP-Complete [Madeira and Oliveira, 2004]. Some of these NP-Complete instances are very close to the instance of definition 2.7, and seem even simpler. Cheng and Church note for example that "the problem of finding the largest square δ-bicluster (s = m) is NP-hard".

From a practical point of view, these results indicate that biclustering algorithms will typically have to work by evaluating all the possible sub-matrices. One can easily show that an expression matrix of size $b * c$ contains $(2^b - b - 1) * (2^c - c - 1)$ sub-matrices of at least two rows and two columns. For an expression matrix with thousands of genes, and dozens of conditions, there are much more than 2^{1000} sub-matrices to test, which is intractable to do in a reasonable time.
For these reasons, most biclustering algorithms use heuristic methods to discover biclusters [Madeira and Oliveira, 2004]. The algorithm proposed by Cheng an Church in [Cheng and Church, 2000] is one of these.

2.2.6 The Cheng and Church's algorithm

Greedy iterative search is a local search meta-heuristic for optimization problems. It starts with a candidate solution picked from the search space and tries to improve it step by step. At each step, a new candidate solution is chosen, which is an improvement of the previous candidate solution. The principle of greedy iterative search is that this improvement is enforced by applying one of a fixed set of possible small modifications to the current candidate solution that improves this candidate solution in a maximal way.

Cheng and Church propose three kinds of greedy algorithms for solving the biclustering problem [Cheng and Church, 2000, Madeira and Oliveira, 2004]. Candidate solutions are sub-matrices $(S, M)'$ of EM'. The "single node deletion" algorithm removes at each step the row or the column of the candidate sub-matrix that gives a maximal decrease of the MSR. The "multiple node deletion" algorithm removes at each step all the rows and the columns for which the sum of the residue of their elements is larger than a given threshold. The "node addition" algorithm adds at each step rows and columns that do not increase the MSR. This last algorithm is also able to add rows whose expression trend is a "mirror image" of the expression trend of the rows in the candidate solution. These mirror or inverted rows are indeed interesting from a biological point of view, but are usually not taken into account by the problem specification as it was detailed in definition 2.7.
In order to find a maximal δ-bicluster, Cheng and Church start with the whole expression matrix as a first candidate solution, and apply successively multiple node deletion and single node deletion (in order to find a δ-bicluster), followed by node addition (which aims at maximizing the size of the discovered bicluster).

Cheng and Church's procedure is purely deterministic. This means that repeating its execution will always return the same solution. In order to find several different maximal δ-biclusters (i.e. representing relations between different groups of genes and conditions), Cheng and Church repeat iteratively their procedure, where, in each iteration, the elements of EM' members of the sub-matrices discovered during the previous iterations are replaced by random numbers. This makes it unlikely that these elements participate in the subsequently discovered solutions.
Nevertheless, this replacement occurs only for the two node deletion steps of the iteration.

As node addition is performed using the actual values of EM', some of the discovered sub-matrices could in fact potentially overlap, i.e. share some of the elements of $(EM)'$ [Madeira and Oliveira, 2004]. This is required as, from a biological point of view, biclusters can typically overlap [Bulcke, 2007]. Nevertheless, the discovery of highly overlapping biclusters using this method is unlikely.
Finally, mean row variance is signaled as a mean of rejecting flat δ-biclusters.

Cheng and Church's approach has been widely evaluated, criticized and improved. Two of the main cited drawbacks of Cheng and Church's algorithm are put forward in [Yang et al., 2003].
First, the greedy iterative search approach is notably prone to be trapped in local optima. If a sub-optimal solution is the best within a region of similar solutions, a greedy algorithm that would explore this region would be attracted by this best solution. Once this solution reached, the greedy algorithm is trapped, as any small modification of the best solution of the region produces another solution of the region, which is, by definition, worse.
Secondly, replacing the elements of EM' members of the already discovered biclusters by random numbers induces "a substantial risk that these random numbers will interfere with the future discovery of biclusters, especially those ones that have overlap with the discovered ones".
Nevertheless, the experimental results produced by Cheng and Church reveal that the algorithm is able to find good quality biclusters.

2.2.7 Testing the algorithm with real data

Cheng and Church propose to test their algorithm by applying it on two real expression datasets, which have already been studied through clustering techniques. The studied datasets are the yeast Saccharomyces cerevisiae cell cycle expression data ("Yeast dataset", 2884 genes and 17 conditions) [Cho et al., 1998] and the human B-cells expression data ("Human dataset", 4026 genes and 96 conditions) [Alizadeh et al., 2000]. Missing values (which represent 12.3% of the matrix for the human dataset) in these datasets are replaced by random numbers. This approach is criticized, notably in [Yang et al., 2003], as it can interfere with the discovery of biclusters. This article proposes an alternative approach, FLOC, where the missing values are notably skipped for the computing of row means, column means and sub-matrix means. At the same time, sub-matrices with a too high proportion of missing values in order for their scores being statistically significant are considered as invalid. A solution for finding several different δ-biclusters without making the already discovered ones by random values is also proposed.
The value of δ for the Yeast dataset was determined using the MSR values of the clusters previously found in these data using clustering techniques. A δ value of 300, close to the lowest of these MSR values, was chosen. This value is very low compared to MSR_{ran} for this dataset whose value is 53000. An intuitive validation of this δ threshold was proposed by computing the MSR score of one million of randomly selected sub-matrix of a given size. This allows tho visualize a good estimation of the probabilistic distribution of the score value and the measure was repeated for several typical sizes. The value of δ for the Human dataset was estimated to 1200, by comparing the range and the variance of the expression values with the ones of the yeast dataset.
The evaluation of the discovered δ-biclusters is notably enforced by visual inspection of bicluster expression graphs similar to the ones we presented previously in this text. Cheng

and Church also produce a comparison of their biclusters with the clusters discovered during the clustering analysis's previously applied to these data.

Some authors (for example in [Bryan, 2005]) propose to evaluate the quality of their approach by applying it on expression data where some existing biological relations between genes and/or conditions are already known by biologists. By comparing the computationally discovered biclusters with the real relations between genes and/or conditions, one could better evaluate the biological relevance and effectiveness of the used technique.

2.3 The SEBI/SMOB evolutionary approach for biclustering of gene expression data

We will now describe the SEBI and SMOB genetic algorithms developed by F. Divina in [Divina and Aguilar-Ruiz, 2006, Divina and Aguilar-Ruiz, 2007] for biclustering of gene expression data. These algorithms are the basis for the genetic algorithm developed in this thesis.

2.3.1 Biclustering of gene expression data as an optimization problem

SEBI and SMOB are developed in the framework of the δ-bicluster model specification, as proposed by Cheng and Church, and which we synthesized at definition 2.7. As we already noticed, the biclustering problem, as specified by Cheng and Church, can be seen as an optimization problem. This optimization problem consist in finding a set of sub-matrices of a given data matrix, such that each sub-matrix optimizes a set of quality criteria [Madeira and Oliveira, 2004], enforcing essentially a coherence requirement. As we express the biclustering problem as an optimization problem, we can solve it using evolutionary computation. We establish thus now formally how the Cheng and Church's formal specification of biclustering given at definition 2.7 can be expressed in terms of the definition and taxonomy of optimization problems given in section 1.2:

Search space definition	The set of all sub-matrices $(S, M)'$ of EM', with at least two rows and two columns.
Objective function(s) definition(s)	The objective functions are the mean squared residue (a positive real valued function to be minimized), the size (a positive integer valued function to be maximized) and the mean row variance (a positive real valued function to be maximized).
Number and nature of quality criterion(s)	The problem is multi-objective. One should fine sub-matrices with a low MSR (typically below a chosen threshold value δ), with a maximal size, and a mean row variance sufficiently high. These objectives are interdependent and notably antagonist, as the MSR of a not perfectly coherent sub-matrix can always be reduced by removing a row or a column, i.e. by reducing its size, and as a flat bicluster is perfectly coherent.

Quality of solution(s) searched for	The problem is multimodal. We want to find several different sub-matrices, with a sufficiently low MSR, a maximal size, and a sufficiently high MRV. By different sub-matrices, we mean sub-matrices that hint biological relations between different sets of genes and conditions. We can qualify this difference between two sub-matrices in terms of overlapping, i.e. the fact that the sub-matrices share more or less elements of EM'. If the group of genes S_1 (respectively S_2) and the group of conditions M_1 (respectively M_2) define the sub-matrix (S_1, M_1) (respectively (S_2, M_2)), then if (S_1, M_1) and (S_2, M_2) share the element EM'_{ij} of EM', it means that gene i of EM' is part of S_1 and S_2, and condition j of EM' is part of M_1 and M_2. If two maximal δ-biclusters overlap too much, they typically represent the same relation between the same group of genes and conditions. But some level of overlapping should be allowed, as different biclusters can typically overlap from a biological point of view.
Nature of search space	The search space is a finite set. This problem is a combinatorial problem. Sub-matrices can be seen as a combination between some of the rows and some of the columns of EM', and the number of possible sub-matrices grows exponentially as the number of row and columns in EM' increases ($\#\text{sub-matrices} = O(2^{\text{number of rows+number of columns}})$). Many results seem to indicate that this problem is a NP-Complete problem.

The biclustering problem, as defined by Cheng and Church, is thus a multimodal, multi-objective, and probably NP-Complete combinatorial optimization problem, with a potentially huge search space, whose structure is potentially totally unknown a-priori. It is thus a perfect candidate to benefit from evolutionary techniques for finding high-quality biclusters.

In the two next paragraphs we detail each of the two genetic algorithms developed by F. Divina to solve this optimization problem. The SEBI (Sequential Evolutionary BIclustering) genetic algorithm is detailed before its multi-objective variant, the SMOB (Sequential Multi-Objective Biclustering) genetic algorithm. These two genetic algorithms are described by showing how they instantiate the general framework of general genetic algorithms we defined at section 1.3.

2.3.2 The SEBI genetic algorithm

The SEBI algorithm uses the general selection scheme for genetic algorithms we defined at section 1.3.2. In this section, we detail the main implementation choices, i.e fitness function, genotypic encoding and genetic operators, population initialization and stopping condition, made in SEBI. We also detail the sequential covering technique used to find several different δ-biclusters.

A Fitness function and selection operator

The fitness function used in SEBI, as detailed below, is approximately a linear combination of the three objective functions of the problem: MSR, size and MRV. SEBI transforms thus the multi-objective biclustering problem into a single-objective problem. As the MSR objective has to be minimized, while the size and MRV objective functions have to be maximized, the global fitness function combines linearly the MSR with the size inverse and the MRV inverse, and has to be minimized.

$$fitness((S,M)') = \frac{MSR((S,M)')}{\delta} + \omega_S * (\omega_r * \frac{\delta}{s} + \omega_c * \frac{\delta}{m}) + \frac{1}{MRV((S,M)')}$$

with

- δ the MSR coherence threshold.
- s the number of rows in the sub-matrix $(S,M)'$.
- m the number of columns in the sub-matrix $(S,M)'$.
- ω_S the size weight, ω_r the row weight and ω_c the column weight, three parameterized linear coefficients to be adjusted.

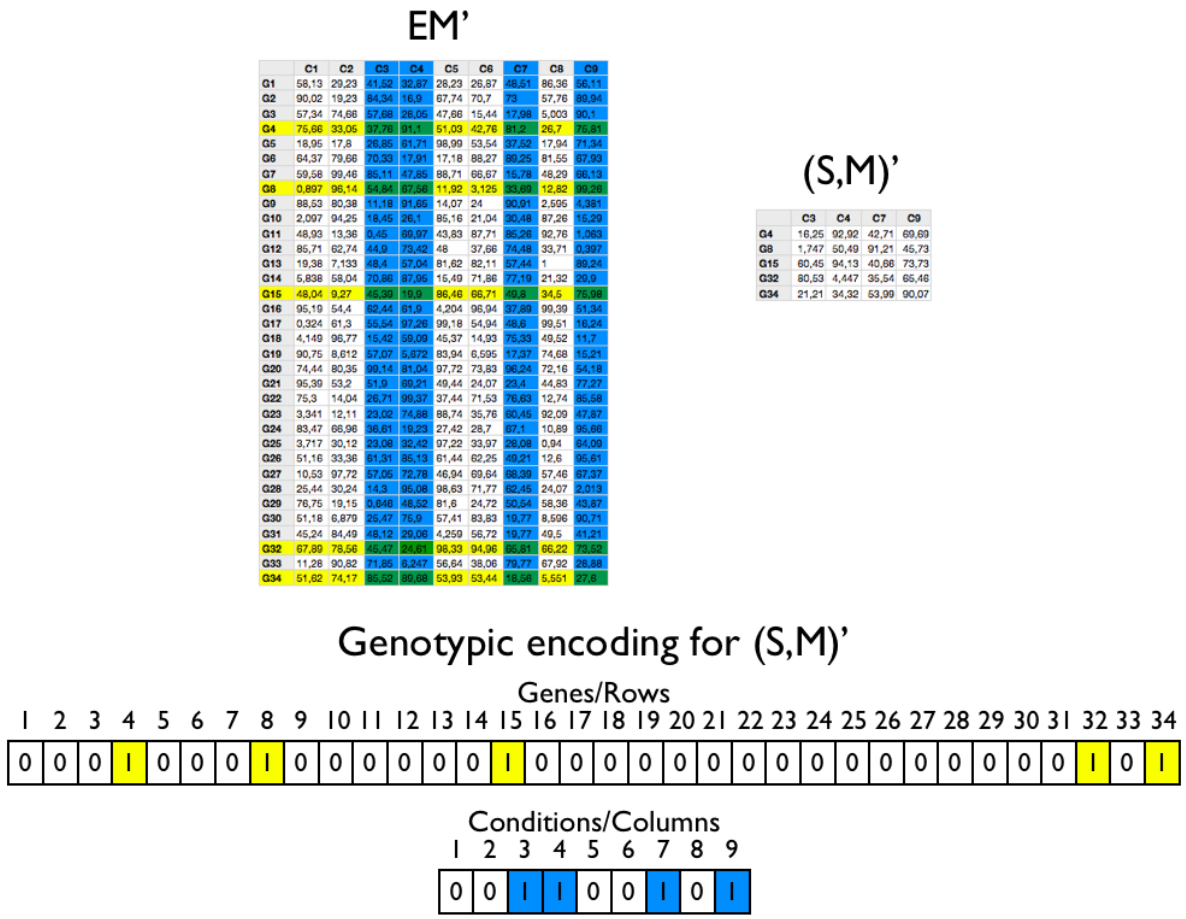
The first term $\frac{MSR((S,M)')}{\delta}$ corresponds to the MSR objective function. It is divided by the constant MSR coherence threshold δ, so that this term returns a value lower than one for δ-biclusters, and larger than one for incoherent biclusters.
The second term $\omega_S * (\omega_r * \frac{\delta}{s} + \omega_c * \frac{\delta}{m})$ corresponds to the size objective function. In fact this not the size of the sub-matrix (i.e. number of rows $*$ number of columns) that has to be maximized directly, but its number of rows and its number of columns separately. The second term of the fitness function can indeed be seen as the sum of the inverse of a number of row objective function $\omega_S * \omega_r * \frac{\delta}{s}$ with the inverse of a number of column objective function $\omega_S * \omega_c * \frac{\delta}{m}$. The parameter ω_S allows to tune the importance of the size criterion for the optimization, compared to the other criteria, MSR and MRV. The parameters ω_r and ω_c allow to define if the maximizing of the size should be enforced by preferring sub-matrices with a large number of rows, or with a large number of columns. The rationale behind this choice is that in the usual expression datasets, the number of genes is very large compared to the number of conditions. Parameters ω_r and ω_c allow thus to balance the fact that s could become very large compared to m.
Finally, the last term $\frac{1}{MRV((S,M)')}$ corresponds to the MRV objective function.

The algorithm uses by default binary stochastic tournament with p=0.9, i.e. the best individual in the tournament has nine chances over ten to be selected.

B Genotypic encoding and variation operators

SEBI uses the fixed length binary string encoding we defined at subsection 1.3.3. The fixed length of the strings equals $b + c$, where b is the total number of rows in EM' and c the total number of columns in EM'. The string can be divided into two parts. The b first bits of the string correspond to the b rows of EM'. The c last bits of the string correspond to the c columns of EM'. A sub-matrix $(S,M)'$ will then be encoded by a string with 0 values everywhere except for the bits corresponding to the rows and columns of EM' that compose this sub-matrix.

The following figure illustrates this kind of encoding. The sub-matrix $(S, M)'$ is composed here of rows 4, 8, 15, 32 and 34, and of columns 3, 4, 7 and 9 of EM'. On can see that in the genotypic encoding of $(S, M)'$ that the bits 4, 8, 15, 32 and 34 of the first part of the string, and also the bits 3, 4, 7 and 9 of the second part of the string have a value 1, while all the other bits have a value 0.



The algorithm applies the three classical binary crossover operators with equal probability: one-point, two-points and uniform crossover. Three mutation operators are also applied in the same way: classical bit-flip operator, and two problem specific operators that respectively add one gene and one column to the sub-matrix. The default rate of crossover used is 0.85 and the default rate of mutation is 0.2.
It should be noted that the previous operators can potentially create genotypes that do not correspond to elements in the search space: genotypes where there is no row/column selected or where there is only one row/column selected.

C Population initialization and stopping condition

The population is initialized with sub-matrices containing only one element of EM', i.e. where only one row and one column are selected. Sub-matrices are then supposed to be "grown" by the algorithm as the generations pass. The size of the population is 200 individuals. The algorithm is stopped after 100 iterations of the generations loop, and the best individual of the last generation is returned if it is a δ-bicluster, otherwise nothing is returned.

D Sequential covering

SEBI does not use a niching method in order to find several different δ-biclusters in the analyzed datasets. Instead, the algorithm is run several times sequentially, and the solution returned by each of the successive runs is one of the different solutions returned by the method. In order for minimizing the overlapping between the δ-biclusters discovered so, a "sequential covering" technique is applied.

For each run of the GA, one can compute the covering value Cov of each element of EM', EM'_{ij}. This covering value is the number of δ-biclusters returned by the previous runs of the GA that contained this element EM'_{ij}. Using these covering values, the penalty of overlapping w_p for each element EM'_{ij} of EM' can be computed in the following way:

$$w_p(EM'_{ij}) = \begin{cases} 0 & \text{if } cov(EM'_{ij}) = 0 \\ \frac{\sum_{y=1}^{b}\sum_{z=1}^{c} e^{-cov(EM'_{yz})}}{e^{-cov(EM'_{ij})}} & \text{if } cov(EM'_{ij}) > 0 \end{cases}$$

with

- b the number of rows in the matrix EM'.
- c the number of columns in the matrix EM'.

This penalty value becomes very large if the considered element of EM' has a large covering value, while many other elements of EM' have a low or zero covering value. As long as the matrix is not uniformly covered, the elements already covered by a discovered δ-bicluster will have a penalty that grows exponentially with their covering value, while the uncovered elements will have a zero penalty.

During the very first run of the GA, the algorithm is run in the way we described in the previous paragraphs. For all the subsequent runs of the GA, the algorithm is run with a modified fitness function, $fitness_{overlapping}$, which increases the fitness of a sub-matrix by the sum of the overlapping penalty of its elements:

$$fitness_{overlapping}((S,M)') = fitness((S,M)') + \sum_{EM'_{ij} \in (S,M)'} w_p(EM'_{ij})$$

With this "sequential covering" mechanism, the sub-matrices that overlap too much with the δ-biclusters discovered previously are strongly penalized during the current run of the GA, compared to the other sub-matrices. The δ-bicluster discovered during one run of the GA will thus typically have a reduced overlapping with the δ-biclusters discovered previously.

2.3.3 The SMOB genetic algorithm

The SEBI genetic algorithm tries to solve the multi-objective biclustering problem by using a linear combination of objectives as fitness, which basically transforms it into a single-objective problem. The SMOB genetic algorithm is presented as an improvement of the SEBI algorithm, which introduces some selection mechanisms based on multi-objective optimization. Moreover, other elements of the GA structure are also different between SEBI and SMOB. In this section, we detail these changes and improvements proposed by SMOB.

A Fitness function and selection operator

The fitness function used in SMOB is partially computed using the Pareto dominance relations and attribute space and phenotypic space distances between the individuals composing the current generation processed by the GA. As we put forward in subsection 1.3.2, the fitness function measures the quality of an individual compared to the other individuals in the current generation of individuals processed by the GA. The fitness function of SMOB approximately measures how much an individual (i.e. a sub-matrix $(S, M)'$) dominates the other individuals in its generation, and how much it is different from these other individuals, using both an attribute space and a genotypic space distance. The more a sub-matrix dominates and is different from the other sub-matrices in the current generation, the more its quality is high, and the more it is favored by selection.

The fact that a sub-matrix $(S, M)'$ dominates more or less the other individuals in its generation P is measured by its dominance strength score $DomStr((S, M)', P)$. The domination relations between the sub-matrices in P are computed using the three objective functions of the biclustering problem: MSR, size and MRV. The dominance strength is the number of individuals of P that $(S, M)'$ dominates, increased by the number of individuals of P dominated by $(S, M)'$ that have a larger MSR than $(S, M)'$. This last element allows to favor the MSR objective compared to the two other objectives.
We measure how much the sub-matrix $(S, M)'$ is different from the other sub-matrices of the current generation P, by using the isolation in the attribute space score $AttIsol((S, M)', P)$ and the isolation in the phenotypic space score $PhenIsol((S, M)', P)$. The isolation in the attribute space score $AttIsol((S, M)', P)$ measures how much a sub-matrix $(S, M)'$ represents a different compromise between the objectives of the problem (MSR, $Size$ and MRV), compared to the other sub-matrices in the current generation P. At each sub-matrix $(S, M)'$ can be associated a 3-dimensional vector

$$\begin{pmatrix} MSR((S, M)') \\ Size((S, M)') \\ MRV((S, M)') \end{pmatrix}$$

inside the attribute space of the problem, $MSR \times Size \times MRV$. We can measure how much two sub-matrices $(S, M)'_1$ and $(S, M)'_2$ represent the different compromises between the objectives of the problem by computing the Euclidian distance between their corresponding vectors in the attribute space:

$$\sqrt{\begin{array}{c} [MSR((S, M)'_1) - MSR((S, M)'_2)]^2 \\ +[Size((S, M)'_1) - Size((S, M)'_2)]^2 \\ +[MRV((S, M)'_1) - MRV((S, M)'_2)]^2 \end{array}}$$

Given a sub-matrix $(S, M)'$ in the current generation, we can measure its Euclidian distance in the attribute space with all the other sub-matrices in the generation P. $AttIsol((S, M)', P)$ is then the shortest of these measured distances. If $AttIsol((S, M)', P)$ is small, it means that there exist at least another sub-matrix in the generation P that represents a similar compromise between objectives as $(S, M)'$. If $AttIsol((S, M)', P)$ is large, it means that the values $(S, M)'$ gives to the objective functions of the problem are very atypic, and that at $(S, M)'$ corresponds to an "isolated" point in the attribute space, within the attribute vectors associated to the sub-matrices in P.

Similarly, the isolation in the phenotypic space score $PhenIsol((S,M)', P)$ measures how much a sub-matrix $(S,M)'$ represents an isolated point in the phenotypic space, i.e. in the search space of the optimization problem, compared to the other solutions in the current generation P. This simply means that $PhenIsol((S,M)', P)$ measures how much $(S,M)'$ is a different solution of the optimization problem, compared to the other solutions that compose the current generation P. $PhenIsol((S,M)', P)$ is thus computed by measuring how much the sub-matrix $(S,M)'$ overlaps with the other sub-matrices in P. Formally, it is defined in [Divina and Aguilar-Ruiz, 2007] as the normalized average of individuals covering the elements of EM' covered by $(S,M)'$.

The sub-matrices $(S,M)'$ having a large dominance strength with large isolation scores in the generation P should be favored by selection. F. Divina proposes thus, by symmetry with the SEBI algorithm, to minimize a fitness that is the sum of the inverses of $DomStr((S,M)', P)$, $AttIsol((S,M)', P)$ and $PhenIsol(S,M)', P)$.
In order to penalize the sub-matrices that are not δ-biclusters, their fitness is increased by the penalty $\frac{MSR((S,M)')-\delta}{\delta}$ that measures how much their MSR is high compared to δ.
The fitness is thus:

$$fitness((S,M)', P) =$$

$$\begin{cases} \frac{1}{DomStr((S,M)',P)} + \frac{1}{AttIsol((S,M)',P)} + \frac{1}{PhenIsol((S,M)',P)} & \text{if MSR((S,M)')} \leqslant \delta \\ \frac{1}{DomStr((S,M)',P)} + \frac{1}{AttIsol((S,M)',P)} + \frac{1}{PhenIsol((S,M)',P)} + \frac{MSR((S,M)')-\delta}{\delta} & \text{if MSR((S,M)')} > \delta \end{cases}$$

The general selection scheme used by SMOB is similar to the one defined in subsection 1.3.2, but SMOB uses elitism. At each iteration of the generations loop, the individuals that compose the Pareto optimal set of the current generation are allowed to survive in the next generation.
The selection operator used is a deterministic tournament operator, with $n = 4$, i.e. where four individuals take part to each tournament.

B Genotypic encoding and variation operators

The genotypic encoding, the variation operators and the crossover/mutation rates used in SMOB are similar to the ones used in SEBI. However, uniform crossover is given an higher probability of use in SMOB than one-point and two-points crossover.

C Population initialization and stopping condition

As in SEBI, the SMOB algorithm is stopped after 100 iterations of the generations loop, but the initialization of the 200 individuals population is different. In order to create each sub-matrix $(S,M)'$ that populates the very first generation of the algorithm, first the number of rows s and the number of columns m of $(S,M)'$ are picked randomly. Then $(S,M)'$ is created by selecting randomly the s genes and m conditions that define $(S,M)'$. The first generation is thus composed of random sub-matrices of random sizes.

D Sequential covering

The SMOB algorithm uses the same sequential covering technique as SEBI to find several different δ-biclusters. However, the $fitness_{overlapping}$ function is defined in a different way:

$$fitness_{overlapping}((S,M)') = fitness((S,M)') + (1 - \frac{size((S,M)') - \sum_{EM'_{ij} \in (S,M)'} cov(EM'_{ij})}{size((S,M)')})$$

Sub-matrices with high sizes that have a low level of overlapping with previously found *delta*-biclusters are favored.

2.3.4 Experimental evaluation of the SEBI/SMOB algorithms

In [Divina and Aguilar-Ruiz, 2006], the SEBI algorithm is evaluated, by applying it to the same Yeast and Human datasets used by Cheng an Church to test their approach (i.e. where missing values have been replaced by the same random numbers), with the same values for δ. Visual inspection of the expression graphs of the obtained results shows interesting "strikingly similar up and down-regulation patterns".
Results are compared to the Cheng and Church's results (CC) [Cheng and Church, 2000] and to the results obtained in FLOC [Yang et al., 2003]. On the Yeast dataset, the CC results and the SEBI results show similar MSR values and FLOC is better. CC does better than SEBI on average in terms of MSR for the Human dataset, but with a much higher standard deviation. The results of CC and FLOC are of larger size than the ones returned by SEBI. This is due to the sequential covering policy used in SEBI. The δ-biclusters discovered during the first runs of the algorithm are large in size but are of bad overall quality. The interesting biclusters are discovered subsequently, but their size is limited by the sequential covering mechanism which prevents them overlapping with the previously discovered biclusters. In CC, the δ-biclusters discovered during the first iterations are also of bad quality, but the following ones can overlap much more in CC than it is allowed in SEBI. Finally, SEBI does better in terms of MRV than CC and FLOC, which means that many of the large size biclusters found by CC and FLOC may not be that much interesting.

In [Divina and Aguilar-Ruiz, 2007], the same evaluation procedure is applied to the SMOB algorithm. SMOB exhibits the same trends as SEBI, while improving its results on the three objectives, MSR, Size and MRV. Moreover, the results discovered by SMOB during the first runs of the GA are directly interesting, in contrast with what happens with CC and SEBI.

2.3.5 Related work: biclustering of expression data using evolutionary computation

Several other authors have applied evolutionary techniques for biclustering of expression data [Gallo et al., 2009].

Notably, a single-objective approach is presented in [Bleuler et al., 2004]. The proposed algorithm is a memetic algorithm, which uses ad-hoc local search techniques inspired by Cheng and Church's algorithm. The EA also uses mechanisms to maintain diversity in the population across the run.

The opportunity to use evolutionary computation for biclustering, even if it requires typically more processing time than greedy methods, is justified by the potential improvement in the quality of the discovered biclusters the searching power of EC could bring. Indeed, "the quality of a biclustering, though, is often considered more important than the computation time required to generate it" [Bleuler et al., 2004].
The single objective fitness function used is the size of the sub-matrix, as the MSR is improved by the ad-hoc local search techniques. The algorithm is only run once and the final generation is returned. The biclusters discovered so are nevertheless said to show a considerable overlap.

[Mitra and Banka, 2006] takes advantage of a state-of-the-art multi-objective evolutionary approach, the Non-Dominated Sorting Genetic Algorithm (NSGA-II, see subsection 1.5.3). The proposed algorithm uses a memetic approach, coupling evolutionary search with local search inspired by Cheng and Church. The objectives taken into account by the approach are the size and the MSR of the considered-matrix. The niching method maintains diversity in the attribute space.
The algorithm is applied on the Yeast and Human datasets and compared to the Cheng and Church's results. The biclusters found are δ-biclusters of larger size those from Cheng and Church's results.

[Gallo et al., 2009] also uses a state-of-the-art multi-objective EA (SPEA2), found to be better than NSGA-II and IBEA for the considered problem, coupled with a Cheng and Church-like local search in a memetic approach. The considered objectives are the number of genes, the number of conditions, the MSR and the MRV of the sub-matrix. The local search approach allows to guide the EA towards the restricted part of the Pareto front where $MSR < \delta$ and to speed up the convergence. The approach also proposes a particular genotypic representation mechanism and genetic operators that allow the EA to search for δ-biclusters containing rows with inverted expression patterns. These rows correspond to the mirror rows that the Cheng and Church's node addition algorithm allows to search for. The local search method used in this memetic algorithm is said to improve Cheng and Church's algorithm by allowing to take into account the row variance of the processed sub-matrices. The algorithm is tested using the Yeast and Human datasets and the obtained results are reported to be better than those of [Mitra and Banka, 2006].

Memetic multi-objective EAs seem thus to constitute the actual trend in evolutionary biclustering techniques. Notably, [Amant, 2010] improves the SMOB algorithm by employing a memetic approach, integrating notably a Cheng and Church-like local search method. Nevertheless, other approaches also exist, like in [Fei and Juan, 2008] (which combines NSGA-II with the Estimation of Distribution Algorithm), and in [Nepomuceno et al., 2010] (which notably proposes a new quality measure for sub-matrices, and proposes a new evolutionary meta-heuristic based on Scatter Search).

Part III

MOBPEOC, presentation and experimental evaluation of a new evolutionary biclustering approach

Chapter 3

Introducing Multi-Objective Biclustering with Probabilistic Encoding and Overlapping Control

Contents

3.1 Introduction: MOBPEOC, a new evolutionary approach for biclustering of expression data

The main contribution of this thesis is to introduce a new evolutionary approach for biclustering of gene expression data. We call it Multi-Objective Biclustering with Probabilistic Encoding and Overlapping Control (MOBPEOC). This new approach is based on the δ-bicluster model. As in the SEBI-SMOB approach, it uses a genetic algorithm that takes into account MSR, $size$ and MRV as objective functions. But the MOBPEOC approach also introduces new features, that define a totally new and potentially improved biclustering process:

- The MOBPEOC approach introduces a new evolutionary technique, called probabilistic encoding. Using probabilistic encoding, the genetic algorithm of the MOBPEOC approach can search for partial combinations of rows and columns of EM', where the presence or absence of a row or column in a bicluster can be specified with a given level of doubt. This can allow the algorithm to deal more intelligently with the conflicting objectives that MSR, $size$ and MRV are, and can increase the search power of the algorithm. Moreover, an important contribution of this thesis is to propose a first life-size test of the probabilistic encoding technique.

- The genetic algorithm of the MOBPEOC approach combines in a new way the fitness sharing state-of-the-art multimodal evolutionary technique, with the niched Pareto selection state-of-the-art multi-objective evolutionary technique. This allows the GA to search for several different partial combinations of rows and columns of EM' that induce low MSR with high MRV and reasonable size. This technique also allows the GA gain some control over the level of overlapping between the individuated different partial combinations.

- When a set of different and interesting partial combinations of rows and columns have been individuated, the MOBPEOC approach allows to exploit these results subsequently in an independent way, in order to find several interesting and different precise biclusters. This research can be enforced according to any a-posteriori policy to favor some particular compromise between MSR, $size$ and MRV. An approach favoring low MSR biclusters is proposed.

In this chapter, we present the MOBPEOC approach in details. In section 3.2 we discuss the probabilistic encoding technique we introduced with MOBPEOC. Then, we detail (section 3.3) our implementation of the niched Pareto genetic algorithm, coupled with the fitness sharing method, which allows to control the level of overlapping between the individuated solutions. Based on the two previous sections, section 3.4 details the general algorithmic structure of the MOBPEOC genetic algorithm. The technique used in MOBPEOC to exploit the partial combinations of rows and columns found in order to individuate interesting biclusters is presented in section 3.5. Finally we conclude this chapter with a very brief description of our technical implementation of the MOBPEOC approach (section 3.6).

3.2 Using a probabilistic encoding

Probabilistic encoding was proposed by F. Divina as an unpublished idea of a new evolutionary technique [Divina, 2008]. In some optimization problems, it is not possible to individuate a precise solution, due to uncertainty over the relevance of some characteristics to be part of the solution or not. Probabilistic encoding is a new kind of genotypic encoding that extends the traditional binary encoding often used in GAs, and which allows to deal with such uncertainty in the problem. This new evolutionary technique has been further developed and tested on the biclustering problem in this thesis.

3.2.1 Binary encoding and uncertainty in optimization problems

In a combinatorial optimization problem, each solution of the problem can typically be seen as a particular combination of some of the members of a finite set of possible traits, like features, properties or elements. In order to find the solution using an exact method, each allowed combination of the possible traits should generally be tried.
For example, in the biclustering problem, each sub-matrix $(S, M)'$ is a particular combination between some of the rows and the columns of the expression matrix EM'. The set of possible traits that allow to define a sub-matrix is thus the set of rows or columns of EM'. Two different sub-matrices cannot share the same exact combination of traits. Combinations involving less than two rows or columns are not allowed.
The traditional binary encoding presented in the framework of common GAs is one of the potential representations that can be used to represent such solutions defined by a particular combination of possible traits. There will be as many bits in the encoding as possible traits in the optimization problem. And each bit in a binary genotype will typically indicate the presence or not of its corresponding possible trait in the represented individual.
For example, in the biclustering problem, the SEBI/SMOB evolutionary approach for the biclustering problem uses the traditional binary encoding of GAs in that way. A binary genotype is here a binary string where each of the bits signals the presence or not of a given row/column of EM' in the represented sub-matrix.

Nevertheless, there exist some optimization problems where the complete set of traits that compose an optimal or sufficiently good solution cannot be entirely established. With these problems, there will always exist indeed a certain degree of uncertainty over the relevance of some of the traits for the quality of the solution.
For example, in a multi-objective problem with highly conflicting objectives like biclustering, removing a particular row or column from a sub-matrix may reduce the *size* and the MRV of the bicluster (i.e. the bicluster may then not be maximal and too flat), but reduce its MSR in the same time (i.e. the bicluster may have more chance to hint a true biological relation between genes and conditions). It can be thus very difficult to say, using only the formal specification of the problem in terms of MSR, *size* and MRV, if this row or column should be part of the solution or not. [Gallo et al., 2009] (see subsection 2.3.5) notes for example that biclustering multi-objective algorithms alone obtain poor results, and should thus be guided towards some particular compromises between MSR, *size* and MRV using a memetic approach.
With a binary encoding, the binary individuals in the population manipulated by the GA can only represent individual solutions, where the presence or not of each of the possible

traits must thus be strictly specified. We say that binary individuals represent exhaustive combinations of the possible traits. In order to deal with uncertainty over some traits, it should be possible for the GA to search also for *partial combinations of traits*, where the presence or not of the possible traits in the combination can eventually be specified with a given level of doubt. This technique would be even more relevant given that GAs are inspired by natural evolution, which do not work on particular individuals, but on interesting and penalizing traits and combinations of traits, which respectively propagate or disappear as the generations pass.

3.2.2 The principle of probabilistic encoding

Probabilistic encoding generalizes the traditional binary encoding to allow the representation of such partial combinations of traits. In binary encoding, each bit indicates the presence or not of its corresponding trait in the represented individual. In probabilistic encoding, we replace each of these bits in the encoding by a real number in $[0, 1]$ that measures the probability of presence associated to the trait in the partial combination of traits represented by the individual.

For example, in the biclustering problem, each probability in a probabilistic genotype is thus associated to one of the genes or conditions of EM', and allows to specify the level of doubt we have on this gene/condition being part of the partial combination of rows and columns represented by the probabilistic genotype. If the probability associated to one gene/condition is close to 1 or 0, the probability simply indicates the presence or absence of the corresponding gene/condition, like in the binary individuals used within SEBI/SMOB. The more the probability is close to 0.5, the more there is a doubt on the presence or absence of the gene/condition in the partial combination of rows and columns represented by the probabilistic individual.

If an exhaustive combination of traits represented by a binary individual defines one solution of the search space, a partial combination of traits represented by a probabilistic individual will define a *region* of solutions of the search space. The solutions that are part of this region will all be the possible solutions of the problem that exhibit those traits whose presence is specified with a small level of doubt in the individual and avoid those ones whose absence is specified with a small level of doubt. The solutions of a region will thus differ by those possible traits whose presence or not is specified with a significant level of doubt in the combination.

The regions defined by probabilistic individuals can have various size. A combination of traits where there is no doubt about any trait represents a region reduced to one solution. In terms of encoding, the probabilities of a probabilistic individual encoding such a region will be close to 0 or 1. The probabilistic individual will thus be reduced to the binary individual representing the only solution in the region. The more the combination is partial, i.e. the more the presence or not of many traits is specified with a high level of doubt, the more the corresponding region is large. A probabilistic individual with all the probability values assuming value 0.5 will represent the whole search space.

One should also note that a GA using probabilistic encoding will thus be able to handle populations of regions of the search space, instead of populations of single solutions. Intuitively, this could improve the exploration power of the GA, especially for problems with huge search spaces, like the biclustering problem.

3.2.3 Evaluating the quality of probabilistic individuals

A GA will use a probabilistic encoding in order to find one or several partial combinations of traits that are particularly pertinent to solve the underlying optimization problem. For example, in the biclustering problem, the discovered partial combinations should thus exhibit high probability values for a group of interesting genes and conditions that define a promising combination, and low probability values for those genes and conditions that would not fit in this combination. At the same time, the genes and conditions whose adequacy is uncertain within the combination should see their corresponding probability be assigned a value measuring the level of doubt about the benefit or penalty they would bring to the combination.

In order for a convergence towards such combinations to occur inside a population of probabilistic individuals, the GA should be able to evaluate how much the partial combination of traits defined by each probabilistic individual optimizes the one or several objective functions of the optimization problem. But an objective function only allows to measure the quality of one single solution of the problem, while a partial combination of traits represent a region of the search space. It seems thus logical to measure the objective function value of a partial combination of traits by measuring the mean objective function value of the region of solutions it represents. In the biclustering problem, the mean MSR, $size$ and MRV of the regions defined by the manipulated probabilistic individuals should be evaluated.
[Divina, 2008] proposes to evaluate the mean value of one objective function in such a region by randomly generating N_{sample} sample solutions representative of this region. The average of the objective function values of these N_{sample} solutions will represent the mean objective function value assigned to the region.
The optimal value for N_{sample} must be evaluated on the basis of the size of the region to evaluate and of the variance of the objective function over this region. A small region with a small objective function variance will require a small N_{sample}, likewise a large region with a large objective function variance will require a large N_{sample}.

Creating a random sample solution representative of the region to be evaluated is very easy. A probabilistic individual associates to each possible trait $\#i$ a probability p_i that measures the level of doubt for this trait to be part of the solution it represents. We will then make each of the possible traits part of the sample solution with a probability that equals this probability p_i.
Concretely, for one possible trait $\#i$, we pick randomly a number nbr_{ran} in $[0, 1]$. The trait $\#i$ will then be part of the sample solution if $p_i \geqslant nbr_{ran}$. By repeating this procedure for each of the possible traits, we define the sample solution by indicating the presence or not of each of the possible traits in this solution, in the same way that binary encoding represents the solutions in general.

3.2.4 Probabilistic encoding as a generic evolutionary technique

In the MOBPEOC approach, we use a GA with probabilistic encoding for biclustering of expression data. An important contribution of this thesis is thus to propose a first life-size test of probabilistic encoding. This technique is indeed not limited to the frame of the biclustering problem, but can be seen as a generic evolutionary technique. It could

be beneficial to all optimization problems that exhibit uncertainty and/or a very large search space.
For example, [Divina, 2008] defines probabilistic encoding as an interesting technique for all problems that necessitate a feature selection process. Feature selection (see, for example, [Liu and Motoda, 1998]) is a typical pre-processing process in machine learning, that allows to "individuate both features that are very important for the problem as well as features that are not useful for the problem" [Divina, 2008]. This allows to discard the uninteresting features from the problem, and to consider only the important features when solving the problem. As a consequence, the performance and scalability of the problem solving processes applied after the feature selection preprocessing is improved. Introducing probabilistic encoding would improve the feature selection process by allowing to deal with features whose relevance cannot be certainly established.

[Divina, 2008] notes that other evolutionary approaches exist to deal with uncertainty in optimization problems, like Estimation of Distribution Algorithms (EDAs) [Lozano et al., 2006] and Fuzzy Genetic Algorithms (FGAs) [Paenke et al., 2006]. FGAs do not deal with uncertainty associated to particular traits. EDAs create a probabilistic model from the best individuals and use it to produce the next generation, without using genetic operators. EDA has already been used, in combination with a state of art multi-objective genetic algorithm, for biclustering of expression data (see subsection 2.3.5).

3.3 Using a niched Pareto genetic algorithm with an overlapping distance

3.3.1 Motivations and principles

As we choose to use probabilistic encoding, the MOBPEOC GA will search for partial combinations of rows and columns of EM' and not for particular sub-matrices. The MOBPEOC GA must thus solve an optimization problem that is slightly different from the one solved in the SEBI/SMOB approach. We can detail this new optimization using the definition and taxonomy of optimization problems given in section 1.2.

Search space definition	The set of all partial combinations of rows and columns of EM' represented by a probabilistic individual, as defined in the previous section.
Objective function(s) definition(s)	The mean MSR, the mean $size$ and the mean MRV of the region of sub-matrices defined by the partial combination of rows and columns. These objective functions are computed using a sample set of sub-matrices representative of the region, as explained in the previous section.
Number and nature of quality criterion(s)	The problem is multi-objective. One should find interesting partial combinations of rows and columns that induce low MSR values and high MRV values without penalizing the $size$.

Quality of solution(s) searched for	As the problem is an extension of the biclustering problem, it is a multimodal problem too. It is indeed particularly important to be able to find several different interesting combinations of rows and columns. At two different discovered partial combinations of rows and columns will obviously correspond two regions composed of sub-matrices sharing a limited number of rows and columns, and thus exhibiting low levels of overlapping. Such different partial combinations will then hint biological relations between different sets of genes and conditions, which is of particular interest for the biclustering problem.

In order to deal with this last multimodal aspect, the MOBPEOC GA uses the state-of-the-art sharing method (subsection 1.4.3) that enforces the discovery of several different interesting regions in one run. The sharing method can be easily combined with a state-of-the-art true Pareto multi-objective selection mechanism for GAs, borrowed from the niched Pareto genetic algorithm (subsection 1.5.3). The combination we propose allows the MOBPEOC GA to deal with both the multimodal and multi-objective aspects of the problem simultaneously in an elegant way, and it offers control over the level of overlapping between the individuated partial combinations of rows and columns.

This niched Pareto genetic algorithm, as proposed in [Horn et al., 1994], is characterized by a selection mechanism based on a Pareto domination evaluation of individuals, combined with the sharing method, but which uses a distance measure between individuals computed in the attribute space. The creation of niches is enforced in the attribute space instead of in the phenotypic space, as in classical sharing, in order to promote a better sampling of the diverse kinds of compromise between objectives that compose the Pareto optimal front of the problem.
In the optimization problem solved here, finding a good sampling of the Pareto optimal front is not really interesting. Most parts of the Pareto optimal front could even be uninteresting, as they could for example be composed of partial combinations exhibiting too high mean MSR values to represent interesting biclusters. The goal that really matters here is to find different interesting partial combinations involving non too overlapping sets of rows and columns. These different combinations could then hint different biological relations between different sets of genes and conditions. It does not really matter whether these combinations represent the same kind of compromise between mean MSR, $size$ and MRV, and whether an individuated partial combination dominates another sufficiently different individuated one.
As a consequence, the MOBPEOC GA will use exactly the same multi-objective selection procedure as the niched Pareto genetic algorithm, but the sharing distance will be computed in the phenotypic space, as in classical sharing. The algorithm should thus be able to establish different niches in the population, corresponding to different interesting partial combinations of rows and columns, and to search for a Pareto-optimal instance inside each niche independently during the run.

The sharing method requires to define a phenotypic distance able to quantify how much two individuals represent different solutions to the problem. For the MOBPEOC problem, such a distance will obviously have to measure to what degree the combinations of rows

and columns represented by two probabilistic individuals are different. We detail the distance used in MOBPEOC in the next subsection. But first, an interesting property of MOBPEOC should be noticed: its potential ability to control the level of overlapping between the different partial combinations of rows and columns it discovers.
Sharing enforces a multimodal optimization by promoting the creation of several niches in the population of the GA, corresponding to different individuated solutions to the problem. In the MOBPEOC problem, there would be one niche for each discovered different combination of rows and columns. The creation of such niches is based on a niching radius parameter. This niching radius controls the minimal distance threshold that must separate two evaluated individuals for the GA to consider them as members of different niches.
By adapting the niching radius we can thus control the level of similarity between the different combinations of rows an columns individuated in each of the discovered niches. Controlling the level of similarity between the different combinations of rows an columns found means controlling the level of overlapping between the sub-matrices the regions defined by these combinations contain, and thus between the potentially interesting biclusters inside these regions. Such an overlapping control can be a breakthrough advantage for the approach, compared to the Cheng and Church or SEBI/SMOB technique, which offer no control and highly limit the allowed level of overlapping, in order to find different biclusters.

3.3.2 Defining a phenotypic distance that measures overlapping

Following the definition of a distance given at subsection 1.4.2, the distance used in the MOBPEOC GA should be zero when applied to a couple of individuals representing combinations involving a same set of rows and columns. The distance should also assume an even larger value that it is applied to a couple of individuals representing combinations involving highly different sets of rows and columns. Finally, this distance should reach a maximal value when applied to a couple of individuals representing combinations involving either totally disjoint sets of rows or totally disjoint sets of columns. Two combinations involving either totally disjoint sets of rows or totally disjoint sets of columns indeed define two regions containing sub-matrices that typically do not overlap.

Let us consider two probabilistic individuals $ProbIndiv_1$ and $ProbIndiv_2$, composed respectively of b probability values $p_1^{row}(i)$ and $p_2^{row}(i)$ with $i = 1...b$ and c probability values $p_1^{column}(j)$ and $p_2^{column}(j)$ with $j = 1...c$, measuring the probability for a row i or a column j among the b rows and the c columns of EM' to be part of the partial combination represented by $ProbIndiv_1$ and $ProbIndiv_2$. We define the rows overlapping ratio $Overlap_{row}$ between $ProbIndiv_1$ and $ProbIndiv_2$ as the ratio between the number of rows of EM' that have a probability $\geqslant 0.5$ to be part of both the partial combinations represented by $ProbIndiv_1$ and $ProbIndiv_2$, and the number of rows of EM' that have a probability $\geqslant 0.5$ to be part of at least one of the partial combinations represented by $ProbIndiv_1$ and $ProbIndiv_2$:

$$Overlap_{row}(ProbIndiv_1, ProbIndiv_2) = \frac{card\{k \in [1,b] | p_1^{row}(k) \geqslant 0.5 \wedge p_2^{row}(k) \geqslant 0.5\}}{card\{k \in [1,b] | p_1^{row}(k) \geqslant 0.5 \vee p_2^{row}(k) \geqslant 0.5\}}$$

We can similarly define the columns overlapping ratio $Overlap_{column}$:

$$Overlap_{columns}(ProbIndiv_1, ProbIndiv_2) = \frac{card\{k \in [1,c] | p_1^{column}(k) \geqslant 0.5 \wedge p_2^{column}(k) \geqslant 0.5\}}{card\{k \in [1,c] | p_1^{column}(k) \geqslant 0.5 \vee p_2^{column}(k) \geqslant 0.5\}}$$

The rows and columns overlapping ratios are basically good estimations of respectively the proportion of shared rows and shared columns between the two combinations of rows and columns defined by $ProbIndiv_1$ and $ProbIndiv_2$. The minimum value for the ratio is zero when the two combinations involve respectively disjoint sets of rows/columns. The maximum value for the ratio is one when the two combinations involve respectively equal set of rows/columns.
We define the matrix overlapping ratio $Overlap_{matrix}$ as the product between the rows overlapping ratio and the columns overlapping ratio:

$$Overlap_{matrix}(ProbIndiv_1, ProbIndiv_2) \\ = Overlap_{row}(ProbIndiv_1, ProbIndiv_2) * Overlap_{columns}(ProbIndiv_1, ProbIndiv_2)$$

The minimum value for the matrix overlapping ratio is zero when the two compared combinations involve either disjoint sets of rows or disjoint sets of columns. The maximum value for the matrix overlapping ratio is one when the two combinations involve both equal sets of rows and columns. The MOBPEOC genetic algorithm can then use a distance, called the *overlapping distance*, $dist_{overlapping}$, defined as:

$$dist_{overlapping}(ProbIndiv_1, ProbIndiv_2) = 1 - Overlap_{matrix}(ProbIndiv_1, ProbIndiv_2)$$

As required, this distance will assume value zero when the two compared combinations involve respectively totally equal sets of rows and columns. It will also assume its maximal value one when the two compared combinations involve either totally disjoint sets of rows or totally disjoint sets of columns. Finally, the distance will be larger as the sets of rows and columns of the compared individuals are different.

3.4 Structure of the MOBPEOC genetic algorithm

In the two previous sections, we have established that the MOBPEOC genetic algorithm had to solve an optimization problem that requires to find a set of different partial combinations of rows and columns of EM', with low mean MSR, high mean MRV without penalizing the mean *size*. In order to solve this problem, we chose to use a probabilistic encoding to represent the partial combinations inside the MOBPEOC GA, and proposed to take advantage of a niched Pareto selection mechanism in this GA, coupled with an overlapping distance.
In this section, we detail the complete algorithmic structure of the MOBPEOC GA and show how it instantiates the general framework of common genetic algorithms we defined at section 1.3.

3.4.1 Generations loop and general selection scheme

The main body of the MOBPEOC GA uses a classical generations loop, whose algorithmic structure is detailed in the following figure. We detail the key processes of this generations loop in the paragraphs below.

```
Population mobpeocGA ()
{

// ———— Parameters ————
maxNumberOfGenerations = ... ;
numberOfIndividualsPerGeneration = ... ;
numberOfReinitializations = ... ;
maxCyclesOfReinitialization = ... ;
// ———— Parameters ————

Population currentGeneration,nextGeneration;

currentGeneration = generatePopulation(numberOfIndividualsPerGeneration);
numberOfGenerations = 0;

while (numberOfGenerations<=maxNumberOfGenerations) {

  if (numberOfGenerations<=maxCyclesOfReinitialization) {

        nextGeneration = generatePopulation(numberOfReinitializations);

        numberOfMissingIndividuals =
                numberOfIndividualsPerGeneration-numberOfReinitializations;

        nextGeneration = nextGeneration
                + createBySelectionReproductionMutation(currentGeneration,
                                                numberOfMissingIndividuals);
  } else {

        nextGeneration = createBySelectionReproductionMutation
                (currentGeneration, numberOfIndividualsPerGeneration);
  }

  currentGeneration = nextGeneration;
  numberOfGenerations = numberOfGenerations + 1;

}

return currentGeneration;

}
```

Each of the individuals in the very first generation of the algorithm are generated according to the following process. A random proportion of randomly selected rows and columns of EM' receive a high probability value randomly picked for each selected row/-column between a minimal threshold $threshold_{up}^{row}/threshold_{up}^{column}$ and 1.0. The remaining rows and columns of EM' receive a low probability value picked for each selected row/column between 0.0 and a maximal threshold $threshold_{down}^{row}/threshold_{down}^{column}$. The values $threshold_{up}^{row}$, $threshold_{up}^{column}$, $threshold_{down}^{row}$ and $threshold_{down}^{column}$ are parameters of the algorithm. For each individual generated, the proportion of selected rows is randomly picked between a minimal and a maximal threshold, and the proportion of selected columns is randomly picked between another minimal and a maximal threshold. This adds eight parameters to the algorithm:

```
// ———— Parameters ————
thresholdUpRow = ... ;
thresholdUpColumn = ... ;
thresholdDownRow = ... ;
thresholdDownColumn = ... ;
minProportionOfSelectedRowsAtInitialization = ... ;
maxProportionOfSelectedRowsAtInitialization = ... ;
minProportionOfSelectedColumnsAtInitialization = ... ;
maxProportionOfSelectedColumnsAtInitialization = ... ;
// ———— Parameters ————
```

The first four parameters allow to control the level of partiality of the combinations created, i.e. the size of the regions associated to the created individuals. It should be remembered here that an individual with many probability values close to 0.5 represents a partial combination that exhibits a high level of partiality and defines a large region of the search space. The last four parameters allow to control the mean size and the size variance of the sub-matrices that compose the regions represented by the created individuals.

The stopping condition of the generation loop is a simple counter, and the loop is stopped when the maximal number of iterations is reached. As a niching method is used, several different solutions are supposed to be maintained in the population, and the whole last generation is thus returned by the algorithm at the end of the run.

In order to promote diversity in the population and to enforce the discovery of potential new niches, we use the reinitialization technique (subsection 1.5.2). At each iteration of the generation loop, a small number of new individuals, generated in the same way as the individuals of the first generation, are introduced in the next generation. The number of new individuals introduced in each generation is controlled by the parameter *numberOfReinitializations*. Reinitialization can be applied only during a fixed amount of the first iterations of the generations loop, controlled by the $maxCyclesOfReinitialization$ parameter. This allows to avoid polluting the very last generation containing the results of the algorithm with pure random solutions.

The *createBySelectionReproductionMutation* function creates and returns the individuals that will compose the next generation, using the general selection scheme of the MOBPEOC GA. The detailed algorithmic structure of the *createBySelectionReproductionMutation* function is available in the following figure. The general selection scheme implemented there is very similar to the one described in the common GA framework. However, one should notice that the selection operator receives both the current generation and the provisional next generation as input. This is necessary as the selection operator will use the continuously updated sharing mechanism of the niched Pareto GA.

```
Population createBySelectionReproductionMutation (currentGeneration,
                                                  numberOfIndividualsToCreate)
{

// ---------- Parameters ----------
reproductionProbabilityRate = ... ;
// ---------- Parameters ----------

Population nextGeneration;

for(int i=1 ; i <= numberOfIndividualsToCreate/2 ; i++ ) {

        father = selectionOperator(currentGeneration, nextGeneration);
        mother = selectionOperator(currentGeneration, nextGeneration);

                if (random(0,1)<reproductionProbabilityRate) {

                        [son,daughter]=reproductionOperator(father,mother);

                        son = mutationOperator(son);
                        daughter = mutationOperator(daughter);

                        nextGeneration.add(son);
                        nextGeneration.add(daughter);

                } else {

                        newFather = mutationOperator(father);
                        newMother = mutationOperator(mother);

                        nextGeneration.add(newFather);
                        nextGeneration.add(newMother);

                }

}

return nextGeneration;

}
```

In the following parts of this section, we detail the structures of the *selectionOperator*, *reproductionOperator* and *mutationOperator* functions. These functions obviously implement the selection and variation strategies used in the MOBPEOC GA.

3.4.2 Niched Pareto selection operator with overlapping distance

The following figure details the algorithmic structure of the *selectionOperator* function. It basically picks one individual in the current generation using the selection operator mechanism from the niched Pareto GA, but using the overlapping distance we defined in the previous section. We detail the key processes of this selection mechanism in the paragraphs below.

```
Individual selectionOperator (currentGeneration, nextGeneration)
{

// ———— Parameters ————
sizeOfComparisonSet = ... ;
nichingRadius = ... ;
scalingFactor = ... ;
// ———— Parameters ————

firstCandidate = pickRandomIndividual(currentGeneration);
secondCandidate = pickRandomIndividual(currentGeneration);

// First step

meanMSR1 = computeMeanMSR(firstCandidate);
meanSize1 = computeMeanSize(firstCandidate);
meanMRV1 = computeMeanMRV(firstCandidate);

meanMSR2 = computeMeanMSR(secondCandidate);
meanSize2 = computeMeanSize(secondCandidate);
meanMRV2 = computeMeanMRV(secondCandidate);

firstCandidateDominated = false;
secondCandidateDominated = false;

for(int i=1 ; i <= sizeOfComparisonSet ; i++ ) {

        comparedIndividual = pickRandomIndividual(currentGeneration);

        meanMSRComp = computeMeanMSR(comparedIndividual);
        meanSizeComp = computeMeanSize(comparedIndividual);
        meanMRVComp = computeMeanMRV(comparedIndividual);

        firstCandidateDominated = firstCandidateDominated ||
                        (meanMSRComp<=meanMSR1 &&
                            meanSizeComp>=meanSize1 &&
                             meanMRVComp>=meanMRV1
                         && !(meanMSRComp==meanMSR1
                                 && meanSizeComp==meanSize1
                                 && meanMRVComp==meanMRV1));

         secondCandidateDominated = secondCandidateDominated ||
                        (meanMSRComp<=meanMSR2 &&
                            meanSizeComp>=meanSize2 &&
                             meanMRVComp>=meanMRV2
                         && !(meanMSRComp==meanMSR2
                                 && meanSizeComp==meanSize2
                                 && meanMRVComp==meanMRV2));

}

if (!firstCandidateDominated && secondCandidateDominated)
        return firstCandidate;

if (firstCandidateDominated && !secondCandidateDominated)
        return secondCandidate;
```

```
// Second step

nicheCount1 = 0;
nicheCount2 = 0;

for individual in nextGeneration {

        distance1 = computeOvelappingDistance(firstCandidate, individual);
        nicheCount1 = nicheCount1+sh(distance1,nichingRadius,scalingFactor);

        distance2 = computeOvelappingDistance(secondCandidate, individual);
        nicheCount2 = nicheCount2+sh(distance2,nichingRadius,scalingFactor);

}

if (nicheCount1<= nicheCount2)
        return firstCandidate;
else
        return secondCandidate;


}
```

The selection mechanism picks two candidate individuals randomly in the current generation, and returns one of them using a two step tournament.

During the first step, the quality of the two individuals is evaluated using the domination relations in the current generation. The mean MSR, $size$ and MRV of an individual are computed using a sample of sub-matrices from the region it represents, as detailed in subsection 3.2.3. This adds the size of the sample sets as a new parameter of the algorithm.

```
// ---------- Parameters ----------
sizeOfTheEvaluationSampleSets = ... ;
// ---------- Parameters ----------
```

As required by Pareto domination, one individual dominates another one if it has a better or equivalent mean MSR, $size$ and MRV, with at least one of these three scores that is strictly better.

If the first step of the tournament leads to a tie, continuously updated sharing is used to compute the niche count of both individuals in the provisional next generation. The overlapping distance defined in the previous section is used for computing this niche count. The one of the two candidates which has the lowest niche count is returned.

3.4.3 Variation operators for probabilistic encoding

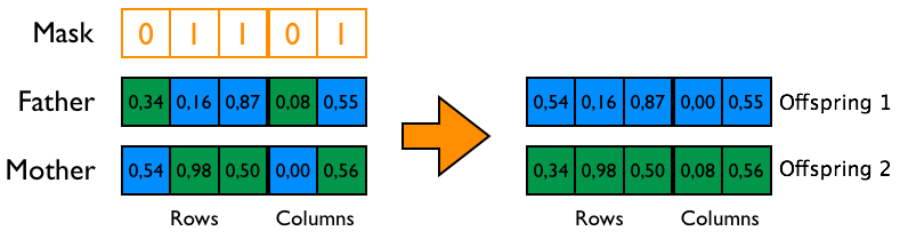
The MOBPEOC GA uses probabilistic encoding to represent individuals. As a consequence, we had to design adapted variation operators for this new kind of genotypic encoding. These new operators have been designed and tested to enforce an effective

exploration of the search space. They are notably inspired by the operators designed for the classical binary representation and by the particular ones used in the SEBI/SMOB approach.

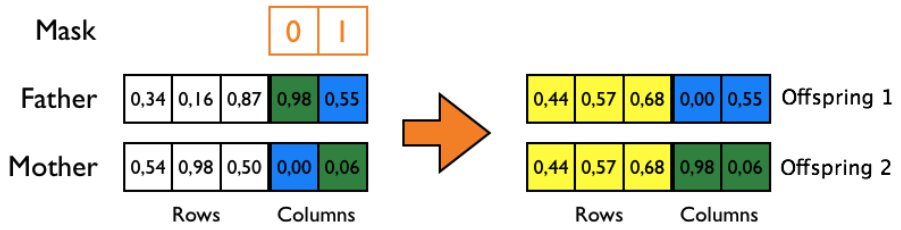
The following figure details the algorithmic structure of the *reproductionOperator* function. Three crossover operators are used: uniform crossover, row mean crossover and column mean crossover. Every time crossover must be applied, one of these three operators is chosen with a parametric relative rate of application. These three operators are detailed in the paragraphs below.

```
[Individual,Individual] reproductionOperator (father, mother)
{

// ---------- Parameters ----------
rowMeanCrossoverRate = ... ;
columnMeanCrossoverRate = ... ;
// ---------- Parameters ----------

randomNumber = random(0,1);

if (randomNumber<rowMeanCrossoverRate)
        return rowMeanCrossover(father, mother);
else if (randomNumber<rowMeanCrossoverRate + columnMeanCrossoverRate)
        return columnMeanCrossover(father, mother);
else
        return uniformCrossover(father, mother);

}
```

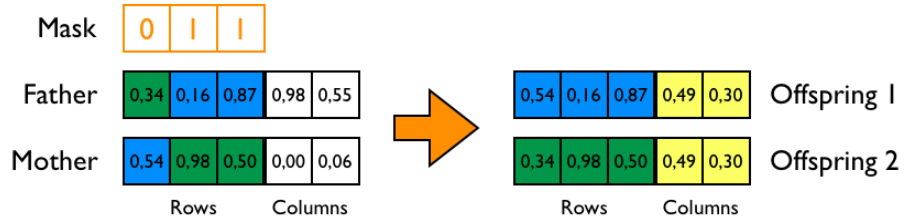
Uniform crossover for probabilistic encoding is illustrated on the following figure. It is inspired and works exactly as the state-of-the-art uniform crossover for binary encoding. Probability values are switched between the two parents at some random positions to create the two offspring.



Row mean crossover (respectively column mean crossover) works exactly as uniform crossover for probability values associated to columns (respectively rows), but for each row (respectively column), the probability values of the offspring equal the mean of the probability values of the two parents. This process is illustrated on the following figures.



Row mean crossover



Column mean crossover

Like in uniform crossover, if both parents have a high probability value for one row (respectively column), the probability will stay high for both offspring. Similarly, if both parents have a low probability value for one row (respectively column), then it will stay low for both offspring. However, if one parent has a high probability value, and the other one a low probability value for the same row (respectively column), it may be interesting to test whether there is uncertainty over the adequacy of the the row (respectively column) within the combination. Contrary to uniform crossover, row mean crossover (respectively column mean crossover) allows to take this fact into account, and it will attribute a probability value around 0.5 to the row (respectively column) for both offspring.
Row mean crossover and column mean crossover work thus as crossover operators, as the allele values of the offspring are derived from the ones of the parents. But they work also as mutation operators as they introduce new allele values, different from the ones of the parents, in the population. This last aspect allows a better exploration by the GA of the very large set of partial combinations of rows and columns.

The following figure details the algorithmic structure of the *mutationOperator* function, which is inspired of the SEBI/SMOB mutation scheme, as it distinguishes mutations for rows and for columns. Two mutation operators are indeed used: row mutation and column mutation. Mutation is applied at a given rate, and every time one chooses to apply mutation on an individual, one of these two operators is chosen with a parametric relative rate of application. Row mutation replace one probability value associated to a row by a random number in $[0, 1]$. Column mutation replace one probability value associated to a column by a random number in $[0, 1]$.

```
Individual mutationOperator (individual)
{

// ---------- Parameters ----------
mutationProbabilityRate = ... ;
rowMutationRate = ... ;
// ---------- Parameters ----------
```

```
if (random(0,1)<mutationProbabilityRate) {

        if (random(0,1)<rowMutationRate)
                return rowMutation(individual);
        else
                return columnMutation(individual);

        }

}
```

3.4.4 Algorithm parameters

We finally synthesize here the twenty-one parameters of the MOBPEOC genetic algorithm. This list will be used as a basis to detail each of the chosen configurations of the algorithm used to produce the experimental results exposed in the next chapter.

```
// Generations loop (2)
maxNumberOfGenerations = ... ;
numberOfIndividualsPerGeneration = ... ;

// Creation of new individuals (8)
thresholdUpRow = ... ;
thresholdUpColumn = ... ;
thresholdDownRow = ... ;
thresholdDownColumn = ... ;
minProportionOfSelectedRowsAtInitialization = ... ;
maxProportionOfSelectedRowsAtInitialization = ... ;
minProportionOfSelectedColumnsAtInitialization = ... ;
maxProportionOfSelectedColumnsAtInitialization = ... ;

// Reinitialization process (2)
numberOfReinitializations = ... ;
maxCyclesOfReinitialization = ... ;

// Selection operator (4)
sizeOfTheEvaluationSampleSets = ... ;
sizeOfComparisonSet = ... ;
nichingRadius = ... ;
scalingFactor = ... ;

// Reproduction operators (3)
reproductionProbabilityRate = ... ;
rowMeanCrossoverRate = ... ;
columnMeanCrossoverRate = ... ;

// Mutation operators (2)
mutationProbabilityRate = ... ;
rowMutationRate = ... ;
```

3.5 Deriving potential biclusters from the discovered partial combinations of rows and columns

3.5.1 Exploiting the results of the genetic algorithm to solve the biclustering problem

After a complete run, the MOBPEOC GA returns its last generation, i.e. a set of probabilistic individuals. As we already stated, each of these individuals represents a potentially different and interesting partial combination of rows and columns of EM'. The rows and columns that have high probability values in the individual define this interesting combination, while the ones with low probability values should be penalizing if added to the combination. Finally, the rows and columns that have average probability values are characterized by a given level of doubt over the interest or penalty to be added to the combination. If the probability value is significant, but smaller than 0.5, then the corresponding row/column should be penalizing for the combination, but with a minimal level of doubt. Similarly, if the probability value is larger than 0.5, but not too close to 1, then the corresponding row/column should be interesting for the combination, but with a minimal level of doubt. If the probability value is around 0.5, then the interest or penalty of the corresponding row/column is totally unknown.
As we already stated, such a discovered partial combination also defines a region of sub-matrices. These sub-matrices share those interesting rows and columns of EM' that define the combination and avoid those rows and columns of EM' that are penalizing for it. They differ by those rows and columns of EM' whose presence in the combination cannot be established undoubtedly by the GA. Among these sub-matrices, the one sub-matrix that contains all the rows and columns that have probabilities larger than 0.5 in the probabilistic individual, and only them, will be the less doubtful interesting compromise solution between MSR, $size$ and MRV of the region. In what follows, we will refer to this solution by the *less doubtful solution*.

Nevertheless, all these results, as interesting as they may be, do not solve the biclustering problem, by returning a set of different biclusters, defining each "a [precise] set of genes showing strikingly similar up-regulation and down-regulation under a [precise] set of conditions". In order to find such precise biclusters in the MOBPEOC approach, we propose to search the region defined by each returned individual, around its less doubtful solution, for a sub-matrix with a low MSR. As the searched bicluster must be a member of a discovered region, and close to its less doubtful solution, it should exhibit a very good MSR value, but stay interesting in terms of $size$ and MRV. Moreover, the GA uses a sharing mechanism to create a set of niches in the population, with a controlled level overlapping between the solutions of each niche. It can thus also be hoped that biclusters representing different solutions to the biclustering problem will be found, by searching each of the regions associated with the individuals of the last generation.

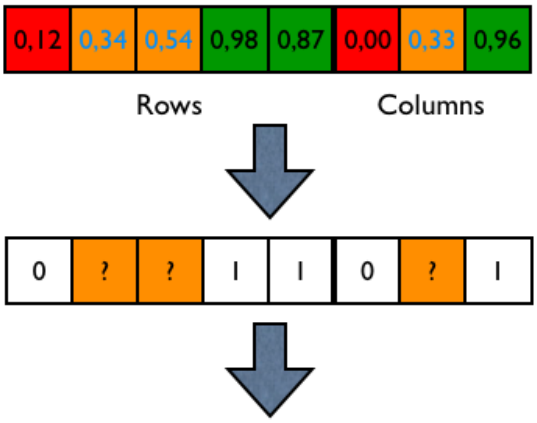
One important advantage of the MOBPEOC approach is that the search phase of the multi-objective biclustering problem, enforced by the GA with probabilistic encoding, is decoupled from the decision making phase, enforced here by the local search around the less doubtful solution for a low MSR bicluster. Several other different decision making processes, favoring other objectives or compromises between objectives, could thus easily be applied to the results of a single search phase. A human decision maker, like a biologist,

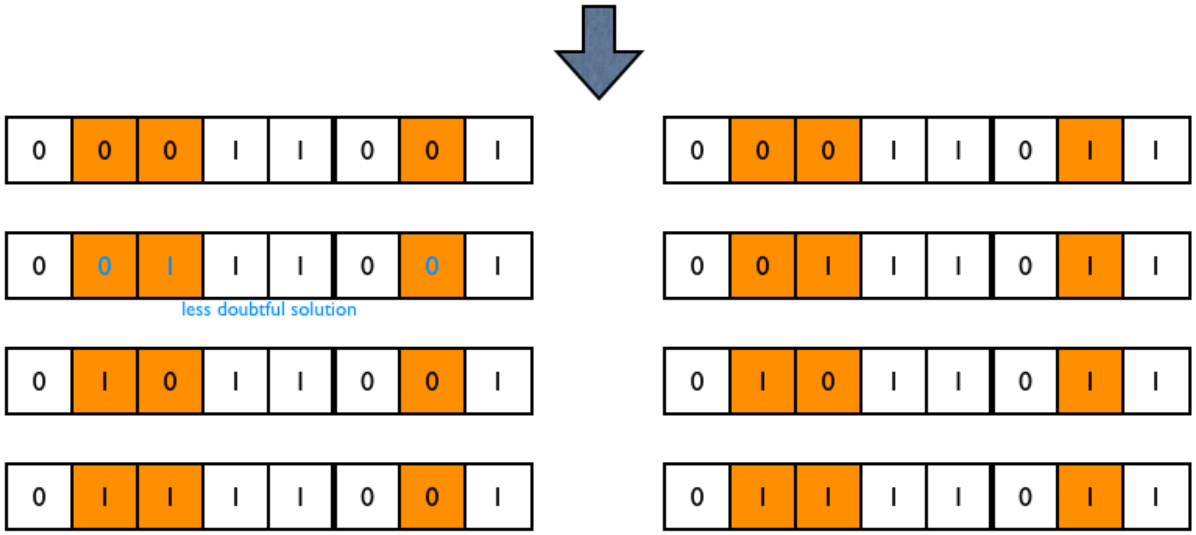
could be able to test different compromises between objectives and choose the best.
In this thesis, we chose to favor biclusters with low MSR in order to demonstrate the ability of the MOBPEOC approach to find highly coherent interesting biclusters. Moreover, starting from these very coherent biclusters, a human decision maker could very easily add or remove some rows or columns to check if the bicluster is maximal and to make it so if necessary. He could be assisted by a computer program that would present him the most interesting rows and columns to add or remove in terms of MSR and MRV. Such an approach could be much more relevant than using an almost arbitrary MSR threshold δ, under which all sub-matrices are considered as coherent biclusters, and above which they are considered as non coherent ones.

In the next subsections, we detail the particular decision making phase we adopted in MOBPEOC.

3.5.2 The MOBPEOC decision making process

In order to be able to search a discovered region around its less doubtful solution for a low MSR bicluster, we should first decide which precise sub-matrices of EM' are part of such a region defined by a probabilistic individual. In MOBPEOC, we choose that all the rows (respectively columns) of EM' that have a corresponding probability smaller than the parametric low probability p_{min}^{row} (respectively p_{min}^{column}) in the probabilistic individual will not be part of the sub-matrices of the region defined by this individual. Similarly all the rows (respectively columns) of EM' that have a corresponding probability larger than p_{max}^{row} (respectively p_{max}^{column}) in the probabilistic individual will be part of all the sub-matrices of the region defined by this individual. All the sub-matrices that do not contain the first set of "forbidden" rows and columns and contain the second set of "mandatory" rows and columns will be part of the region defined by the probabilistic individual.
If we represent each sub-matrix by the set of rows and columns of EM' it contains, using the binary encoding defined in the SEBI/SMOB approach, it appears that all the sub-matrices of a region will be represented by binary individuals where the bits assume value 0 (respectively value 1) if the corresponding probability in the probabilistic individual has a value smaller than p_{min}^{row} for a row and p_{min}^{column} for a column (respectively larger than p_{max}^{row} for a row and p_{max}^{column} for a column). At each possible combination of values for the remaining bits will correspond a binary individual representing one of the sub-matrices of the region, and all the sub-matrices of the region will be represented by such an individual. This is illustrated in the following figure (where we use $p_{min}^{row} = p_{min}^{column} = 0.15$ and $p_{max}^{row} = p_{max}^{column} = 0.85$):





At the probabilistic individual on the top of this figure will correspond the region containing the sub-matrices defined by the binary individuals on the bottom of the figure. The less doubtful solution of this region is tagged with a blue label.
The MOBPEOC decision making process introduces thus the four p_{min} and p_{max} values as new parameters of the approach:

```
// Decision making process (4)
pMinRow = ... ;
pMaxRow = ... ;
pMinColumn = ... ;
pMaxColumn = ... ;
```

In the MOBPEOC decision making process, we propose to use a local search optimization method, that will start with the less doubtful solution of the region, and will search the solutions close to it in the region, to improve its MSR. Practically, in the previous figure, the local search will consider first the solution with the blue label, and try to add or remove it a limited number of the doubtful rows or columns that differentiate the sub-matrices in the region (i.e. the ones with orange corresponding bits in the figure), in order to reduce the MSR.
Our experiments have shown that the size of the set of sub-matrices to search in the decision making phase of the MOBPEOC approach could still be very large. As a consequence, using an exact optimization method to enforce the search is not convenient, and a local search optimization heuristic should be used. A notable candidate is the greedy search approach of the Cheng and Church's algorithm. However, this approach is notably prone to be trapped in a local optimum. As a consequence, MOBPEOC uses the simulated annealing local search optimization meta-heuristic, which is often presented as an improvement of greedy search that allows to overcome the problem of local optima. The superiority of simulated annealing over Cheng and Church's greedy search method for biclustering of expression data is notably suggested by experimental results obtained in [Bryan, 2005]. In the two next subsections, we present respectively the simulated annealing general meta-heuristic technique, and its implementation in the MOBPEOC approach.

3.5.3 The simulated annealing local search meta-heuristic

Simulated annealing (SA) [Kirkpatrick et al., 1983, Černý, 1985] is an optimization meta-heuristic inspired by the physical field of statistical thermodynamics. In this subsection, we detail the generic algorithmic structure of the basic implementation of SA for single-objective optimization. We also present the main issues for developing an effective SA for a particular optimization problem. Details on historical and theoretical aspects of simulated annealing, as well as on its advanced variants can be notably found in [Dréo et al., 2003, Chapter 1], and the many references therein.

As any local search method, simulated annealing sequentially considers one potential solution at a time, jumping from one solution to another in the search space, until a solution considered as optimal is discovered or a limit of computing time is reached. The particularity of simulated annealing holds in the way the algorithm jumps from one solution to another solution to explore the search space.
The SA algorithm is a stepwise process. At each step is associated a different value of temperature (remember that SA is inspired from thermodynamics). The temperature of the first step is a parameter of the algorithm, and it is reduced by a given parametric ratio in each step of the algorithm. For example, if the initial temperature in the first step is 10, and the decrease ratio is 0.5, then the temperature of the second step will be 5, the one of the third step will be 2.5 and so on.
When SA tries to jump from its current solution to a new one in the search space, it creates a candidate next solution by applying a random elemental modification to the current solution. The objective function values of both the current and the candidate solutions are compared, and if the candidate solution is the best, SA jumps to this candidate solution that becomes the new current solution. Otherwise, the candidate solution is not systematically rejected. In fact, it is then accepted with a probability p_{accept}, with

$$p_{accept} = e^{\frac{-|objectiveFunction(candidate) - objectiveFunction(current)|}{temperature}}$$

This probability of acceptation decreases exponentially with the ratio between the objective function penalty if we switch from the current to the candidate solution and the current temperature value in the algorithm. If the candidate solution is accepted, it becomes the current solution, otherwise the current solution is not modified. From the resulting current solution, a new candidate solution is computed and a new trial process is repeated.
During the very first step of the algorithm, the temperature in the algorithm is high. This typically induces that even the candidate solutions with a relatively bad objective function value compared to the current solution have a significant probability p_{accept} to be accepted as new current solution. After a parametric number of trial processes, the algorithm switches to next step and the temperature is reduced. As the algorithm goes through new steps, the more the temperature is reduced and thus the more the objective function penalty must be small for a candidate solution to have a significant probability p_{accept} to replace the current one. The algorithm is stopped after a step in which the temperature has reached a parametric threshold low enough for all the candidates that degrade the objective function to be rejected.
The algorithm can thus switch from a high temperature situation, where any random modification of the current solution that is not too penalizing is accepted, to a low temperature situation where only the modifications that improve the current solution are

accepted. The first situation correspond to a global exploration of the search space, which tries to explore all the parts of the search space to locate an interesting part. The second situation corresponds to a local exploration of the interesting located part of the search space.
Greedy search may be easily trapped by a local optimum, because it cannot switch to a solution that is worse than the current one. As this is allowed and controlled in simulated annealing, the algorithm is much less prone to be trapped by such local optima.

In the next figure, we synthesize the algorithmic structure and the four parameters of a basic simulated annealing algorithm for a single-objective optimization problem (the objective function should be maximized), as we have just detailed it.

```
// ---------- Parameters ----------
initialTemperature = ... ;
finalTemperature = ... ;
numberOfAttempsPerTemperature = ... ;
temperatureDecreaseRatio = ... ;
// ---------- Parameters ----------

temperature = initialTemperature;
currentSolution = selectInitialSolution(searchSpace);

while (temperature > finalTemperature) {

        for (int i=1; i<= numberOfAttempsPerTemperature; i++) {

                candidateSolution
                        = makeRandomElementalChange(currentSolution);

                if (objectiveFunction(candidateSolution)
                                >objectiveFunction(currentSolution)) {

                        currentSolution = candidateSolution;
                }
                else if (random(0,1)<=
                e^((-(objectiveFunction(currentSolution)-objectiveFunction(candidateSolution)))/temperature)) {

                        currentSolution = candidateSolution;
                }
        }

        temperature = temperature * temperatureDecreaseRatio;

}

return currentSolution;
```

As any meta-heuristic, simulated annealing can be instantiated to solve many optimization problems. The main difficulties when instantiating SA for a particular problem consist in defining efficient values for the parameters of the algorithm, and efficient mechanisms to represent and make elemental modifications to the solutions in the search space. Standard answers are commonly cited to solve these problems [Dréo et al., 2003, Pages 40-41].
First the chosen objective function should essentially measure the quality of a solution.

Some penalties can be added to the objective function to drive the search towards some unpenalized particular kinds of solutions, but the exploration should essentially be driven by the set of elemental modifications allowed to be applied to the current solution.
The penalty or improvement of the objective function induced by an elemental change in a solution should be easy to compute. Ideally, it should be possible to evaluate this penalty or improvement as a function of the modification, i.e. without computing the objective function of the candidate solution from scratch.
The initial temperature can be evaluated by computing the mean change in objective function induced by a number of random elemental modifications of some randomly chosen solutions. The initial temperature is then chosen to adjust the probability of acceptation of a candidate solution that would induce a penalty equal to this mean objective function change. This probability of acceptation of an average penalizing candidate solution should depend on the quality of the very first solution considered by the algorithm. According to [Dréo et al., 2003], if this very first solution is considered to be good, the initial temperature should be chosen to adjust the probability of acceptation to ≈ 0.20, otherwise, an initial temperature adjusting a probability of acceptation of ≈ 0.50 should be chosen. The number of attempts for each temperature step can be evaluated to $100 * N_{parameters}$, where $N_{parameters}$ is the number of parameters that individuate a particular solution of the problem. The temperature decrease ratio should assume a value around 0.90. The final temperature can be established by running the algorithm on the optimization problem. If after three successive steps, the current solution is almost never modified, the temperature of the third step is chosen as final temperature.

3.5.4 The MOBPEOC simulated annealing algorithm

The MOBPEOC simulated annealing algorithm is an instance of the generic SA presented in the previous subsection, which allows to search for a low MSR bicluster among the sub-matrices around the less doubtful solution of a given promising region. We review the main particularities of this instance of SA in this subsection.

The objective function of the MOBPEOC SA will simply be the MSR and the very first current solution will be the less doubtful solution of the explored region. Given a current solution, i.e. a sub-matrix $(S, M)'$ member of the explored region, the possible elemental modifications to create a new candidate solution that must also be a member of the region will be:

- Add to $(S, M)'$ one of the doubtful rows of EM' differentiating the solutions of the region and that $(S, M)'$ does not already contain.
- Remove from $(S, M)'$ one of the doubtful rows of EM' differentiating the solutions of the region and that $(S, M)'$ already contains.
- Add to $(S, M)'$ one of the doubtful columns of EM' differentiating the solutions of the region and that $(S, M)'$ does not already contain.
- Remove from $(S, M)'$ one of the doubtful columns of EM' differentiating the solutions of the region and that $(S, M)'$ already contains.

This set of possible modifications is particularly natural and allows any solution of the region to be reached by the SA starting from the most representative solution of this

region. During the run of the GA, it will not be necessary to compute the MSR of each new candidate solution from scratch. One can indeed check that with the set of possible modifications we defined, the MSR of a new candidate solution can typically be computed by slightly modifying a part of the computations made for the current solution.
As we only want to locally explore the part of the region around the less doubtful solution, a relatively low initial temperature should be chosen, but nevertheless sufficiently high to prevent the algorithm from being trapped in a local optimum.
As a bicluster will typically contain much more rows than columns, adding or removing a column to a current sub-matrix solution will usually add or remove much more elements from the sub-matrix than adding or removing a row. The MSR of the current solution is thus generally more modified when the candidate solution is created by modifying its set of columns than its set of rows. As a consequence, the MOBPEOC SA uses two temperatures that will be reduced simultaneously at each temperature step: a row temperature and a column temperature. When a modification involves a row, the probability of acceptance is computed using the row temperature, and when it involves a column, it is computed using the column temperature.
The number of attempts for each temperature step will be the number of doubtful rows and columns in the explored region. This value is small compared the one advised in the previous subsection, as $N_{parameters}$ is the number of doubtful rows and columns in the explored region. Using such a small value allows to prevent an extension of the search to solutions too far from the less doubtful solution of the region. The temperature decrease ratios and the final temperatures can be adjusted as detailed in the previous subsection.

We finish this section by synthesizing the new parameters introduced by the SA in the MOBPEOC approach:

```
// Temperature (4)
initialRowTemperature = ... ;
initialColumnTemperature = ... ;
finalRowTemperature = ... ;
finalColumnTemperature = ... ;

// Other parameters (2)
numberOfAttempsPerTemperature = ... ;
temperatureDecreaseRatio = ... ;
```

3.6 Code implementation

An important part of the work made in this thesis was to test the MOBPEOC approach on real expression data. The obtained results are presented in the next chapter. For these tests, both the MOBPEOC GA and SA have been implemented using the Java programming language. The main goal while developing the code was not to produce a high performance production code, but to create an experimental prototype to test different evolutionary techniques and parameter configurations. The code should thus be able to support frequent changes in a robust way, and to integrate many logging mechanisms. The Java platform, with the modularity of its object-oriented paradigm, and its existing aspect-oriented programming framework (AspectJ) allowing to weave

logging aspects in the "business" code on demand and without modifying it, was used to enforce efficiently such an experimental approach.
Parts of the programs, like the creation of the next generation in the GA or the search of the different discovered regions using SA, were parallelized using a Thread Pool mechanism. This allowed to reduce by an important ratio the amount of time necessary to execute the different algorithms on the bi-processor machine used for the tests. For the creation of a next generation in the GA using the general selection scheme, some synchronization mechanisms had to be added to the parallel implementation. This is due to the fact that the continuously updated sharing mechanism supposes that the creation of the individuals in the next generation is sequential, in order to be able to compute the niche count of the candidate individuals in the provisional created new generation.

Chapter 4

Experimental results and discussion

Contents

4.1 Introduction: experimental methodology

In order to test the efficiency of the MOBPEOC approach on real expression data, our Java implementation has been run over the well known Yeast and Human gene expression datasets (subsection 2.2.7). This allows us to compare the results obtained using MOBPEOC with the ones returned by the original biclustering techniques (Cheng and Church algorithm, FLOC), and with the SEBI/SMOB evolutionary biclustering techniques, which inspired MOBPEOC. The datasets used here were the same as the log-transformed ones used in the experiments lead with the Cheng and Church, SEBI and SMOB algorithms (i.e. where the missing values have been replaced by the same random numbers).
The experimental testing process involved essentially three different activities:

1. **Optimization of the parameters within the overall approach**. As the architecture of the MOBPEOC approach was experimentally designed and tested, its large number of parameters were adjusted, in order to induce the most efficient discovery of different interesting partial combinations of rows and columns. The parameters were adjusted to the Yeast dataset, then subsequently adapted to the Human dataset.

   The parameters of the GA were adjusted using parameter tuning. The parameter configuration used in the SEBI/SMOB approach and the parameter values commonly cited in the literature were used as the starting point for this parameter tuning process. The quality of the obtained set of parameter values seemed sufficient, so that using a more advanced technique than parameter tuning to adjust the parameters seemed unnecessary. The parameters for the decision making process were adjusted experimentally to obtain interesting biclusters. The parameters of the SA were adjusted according to the rules detailed in the previous chapter.

2. **Collecting, statistical analysis and visualization of the results and of execution informations**. The algorithm was run over each of the two datasets. During and after each run, many informations over the search process and the obtained results were logged, aggregated through several statistical techniques, and effectively presented through adequate visualization methods. These output data were used both for decision-making while designing and adjusting the parameters of the MOPBEOC approach, and for the subsequent final analysis of the biclusters returned by MOBPEOC.

3. **Interpretation of the results and of the search process, and comparison with other techniques**. For each dataset, the final results, statistics and graphs were analyzed, interpreted and compared with the outputs of the other considered biclustering techniques.

In this chapter, for each of the two datasets (sections 4.2 and 4.3), we present the parameter configuration established for the effective run of MOBPEOC over the dataset, and we discuss the efficiency of the method, based on the obtained results, statistics and graphs, and on their comparison with the outputs of the other biclustering techniques.

4.2 Biclustering of the Yeast dataset

4.2.1 Configuration of the MOPBEOC parameters

We detail and comment here the parameter values that were used within MOBPEOC for the biclustering of the Yeast dataset.

A Configuration of the genetic algorithm

```
// Generations loop (2)
maxNumberOfGenerations = 1000 ;
numberOfIndividualsPerGeneration = 600 ;
```

In SEBI/SMOB, a population of 200 individuals was used and evolved during 100 generations. Using a larger population size in the MOBPEOC GA allows to establish many niches within the population and thus to discover many different interesting partial combinations of rows and columns in one run. A larger population typically requires a larger total number of generations to converge to interesting parts of the search space, and thus a larger computing time. The chosen values allows a good compromise between discovery of many different partial combinations and the required computing time.

```
// Creation of new individuals (8)
thresholdUpRow = 0.7 ;
thresholdUpColumn = 0.7 ;
thresholdDownRow = 0.3 ;
thresholdDownColumn = 0.3 ;
minProportionOfSelectedRowsAtInitialization = 0.001 ;
maxProportionOfSelectedRowsAtInitialization = 0.04 ;
minProportionOfSelectedColumnsAtInitialization = 0.7 ;
maxProportionOfSelectedColumnsAtInitialization = 1.0 ;
```

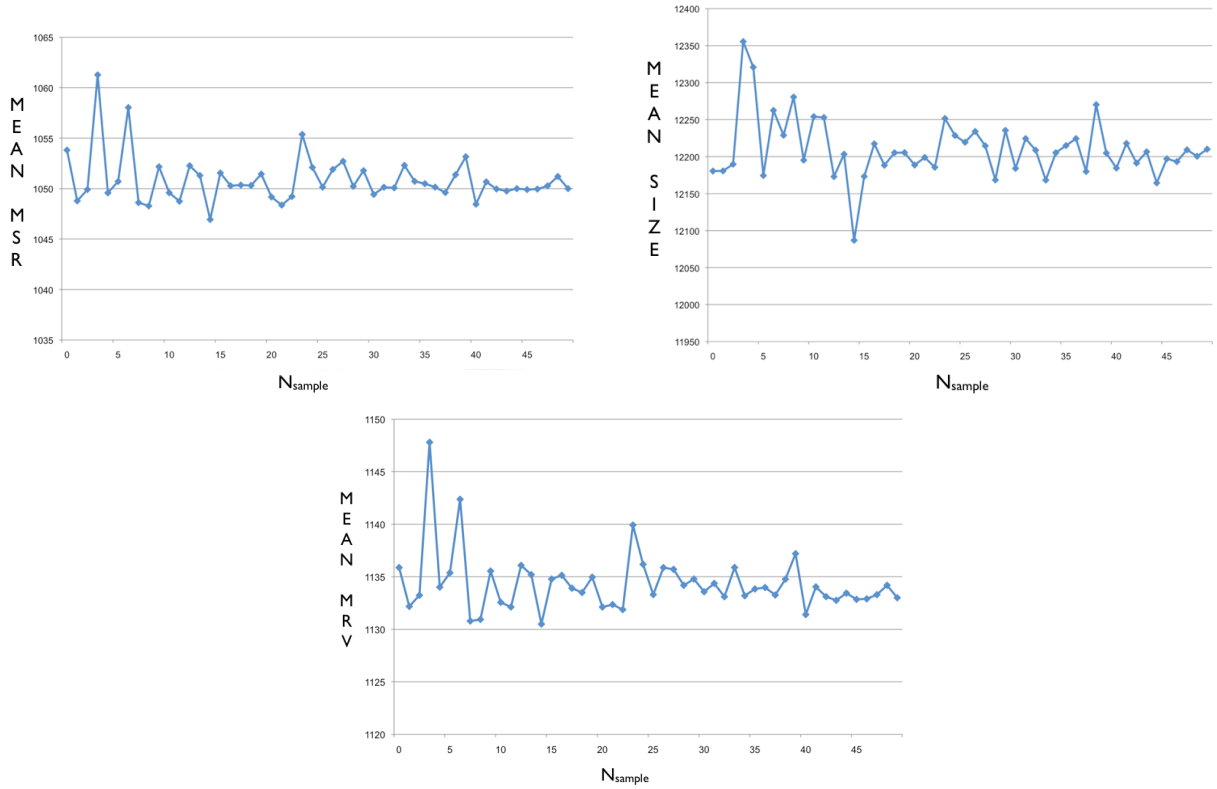
The threshold values chosen here allow to create probabilistic individuals representing regions of reasonable size. The proportion values allow the sub-matrices in these regions to contain a number of rows an columns of the order of the ones of the biclusters discovered using the Cheng and Church and SEBI/SMOB approaches.

```
// Reinitialization process (2)
numberOfReinitializations = 30 ;
maxCyclesOfReinitialization = 990 ;
```

5% of the individuals in a generation are reinitialized individuals. This low proportion allows to enforce a better diversity and a discovery of new niches across the generations, without disrupting the search power of the GA. The ten last generations are not concerned by reinitialization in order to avoid polluting the very last generation containing the results of the algorithm with pure random solutions.

```
// Selection operator (4)
sizeOfTheEvaluationSampleSets = 50 ;
sizeOfComparisonSet = 510 ;
nichingRadius = 0.85 ;
scalingFactor = 1 ;
```

The size of the sample sets was established by computing the objective function values for a population of 200 random probabilistic individuals with different sample set sizes. For each sample set size N_{sample}, the mean of each of the objective function values for the 200 individuals is reported in the following graphs:



One can see that for a sample size of 50, the objective functions values seem to stabilize around an average value, which indicates that this size should be sufficient to estimate these values.
We chose to use a comparison set whose size is 85% of the size of the population in the GA. This is very large compared to the value proposed in [Horn et al., 1994]. This can typically be explained by the fact that the context of use of the Niched Pareto technique is very different in MOBPEOC, and our tests show that smaller values prevent the population to converge to interesting parts of the search space.
The niching radius assumes value 0.85, which more or less means that the different partial combinations found should present at most 15% of similarity.
The scaling factor assumes value 1, which is the typically cited value in the literature [Dréo et al., 2003, Page 189].

```
// Reproduction operators (2)
reproductionProbabilityRate = 0.85 ;
rowMeanCrossoverRate = 0.025 ;
columnMeanCrossoverRate = 0 ;
```

Due to the small number of columns in the Yeast dataset, no column mean crossover operator is used. An uniform crossover coupled with a standard mutation mechanism is indeed sufficient for a proper exploration of the set of possible combinations of columns. Our experiments have shown that introducing a column mean crossover operator made the variation scheme too disruptive, and prevented a proper convergence of the population.

The global crossover rate used in MOBPEOC is the same that the one used in SMOB, as it seemed effective and coherent with the litterature [Dréo et al., 2003, Page 110]. The rate of row mean crossover was tuned to a value allowing an effective but not too disruptive exploration of the set of possible combinations of rows.

```
// Mutation operators (2)
mutationProbabilityRate = 0.05 ;
rowMutationRate = 0.6 ;
```

The mutation rates were tuned to allow an effective but not too disruptive exploration of the search space. The global rate of mutation is much smaller than in SMOB and closer to the traditional values cited in the litterature [Dréo et al., 2003, Page 90].

B Configuration of the decision-making process

```
// Decision making process (4)
pMinRow = 0.15 ;
pMaxRow = 0.85 ;
pMinColumn = 0.15 ;
pMaxColumn = 0.85 ;
```

The probability values were tuned in order for the regions defined by a probabilistic individual to contain the best panel of solutions representative of the partial combination represented by the individual. Concretely, these adequate probability values should allow and were thus selected to promote a quicker and more efficient discovery of interesting low MSR biclusters, close to the less doubtful solution of the region.

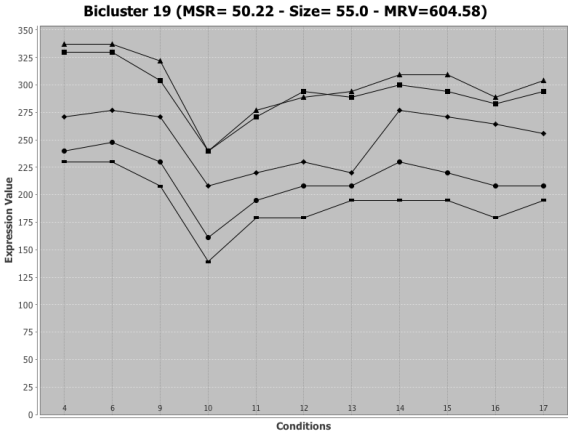
C Configuration of the simulated annealing algorithm

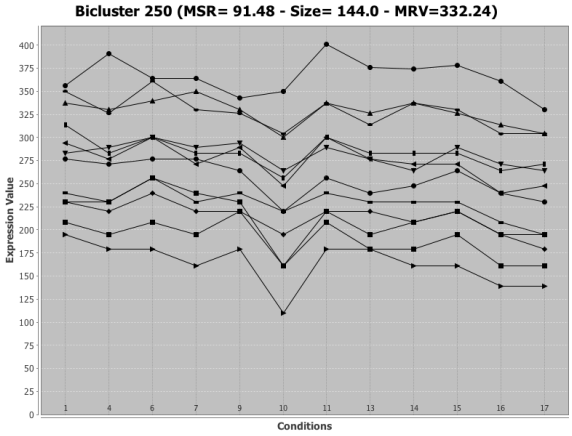
```
// Temperature (4)
initialRowTemperature = 0.08 ;
initialColumnTemperature = 20 ;
finalRowTemperature = 0.015 ;
finalColumnTemperature = 3.75 ;

// Other parameters (2)
numberOfAttempsPerTemperature = number of doubtful rows
                                + number of doubtful columns ;
temperatureDecreaseRatio = 0.90 ;
```

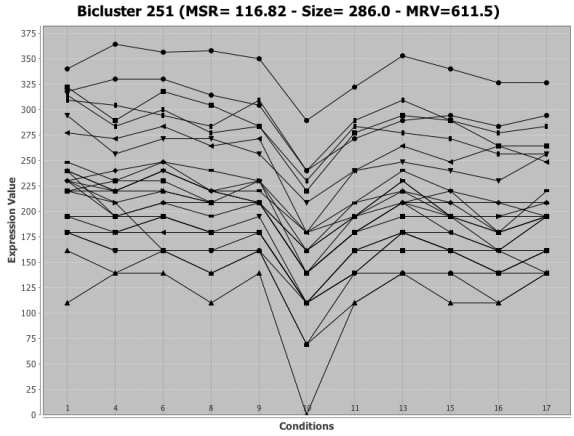
These values have been chosen using the rules detailed in subsection 3.5.4.

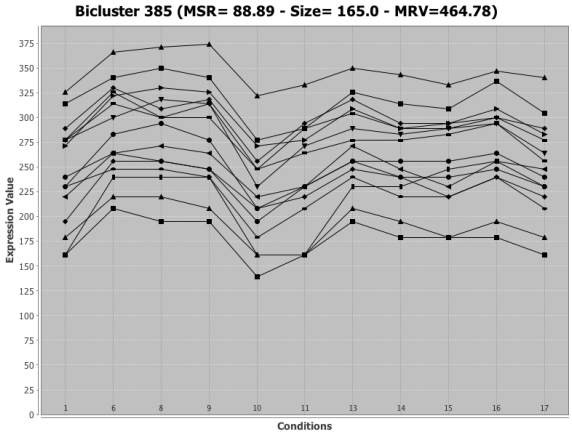
4.2.2 Quality of the discovered biclusters

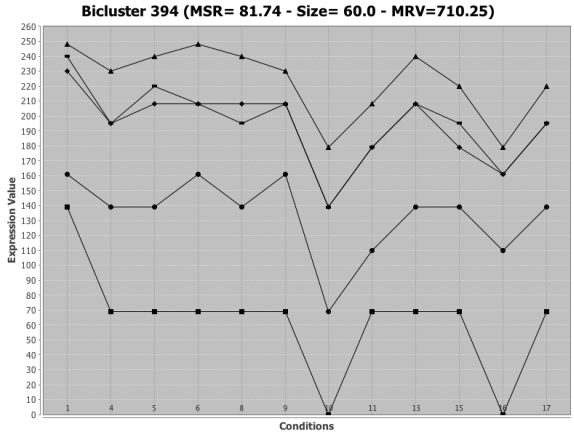
In this subsection, we show and comment the expression graphs of a panel of biclusters extracted from the 600 biclusters (one for each probabilistic individual returned by the GA) returned by the execution of MOBPEOC over the Yeast dataset. The rows and columns that compose the biclusters exposed here are detailed in appendix A.

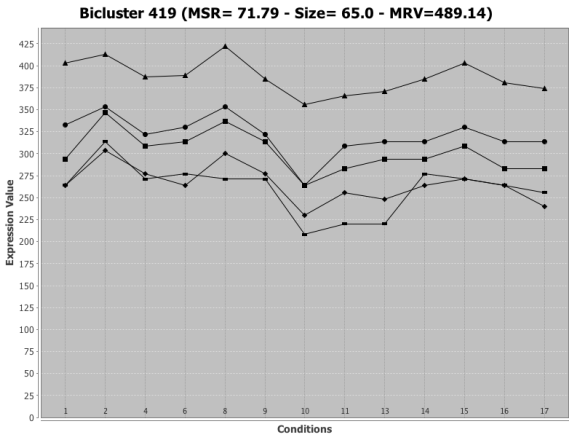

Bicluster 19 (MSR= 50.22 - Size= 55.0 - MRV=604.58)
Expression Value
Conditions

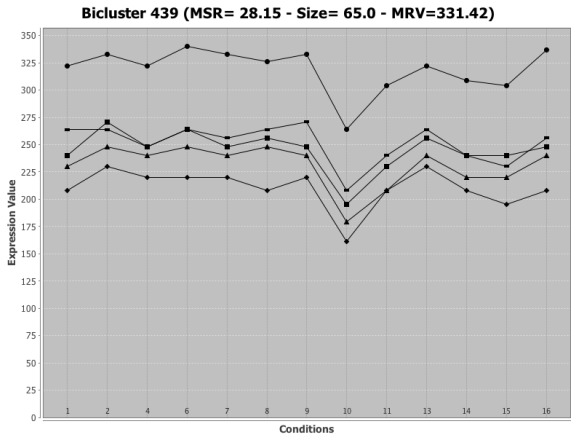

Bicluster 250 (MSR= 91.48 - Size= 144.0 - MRV=332.24)
Expression Value
Conditions

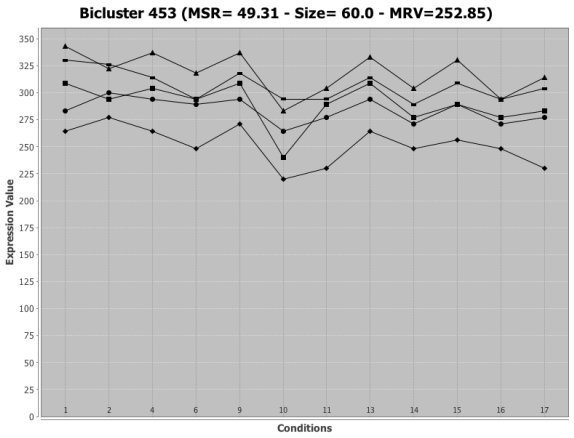

Bicluster 251 (MSR= 116.82 - Size= 286.0 - MRV=611.5)
Expression Value
Conditions

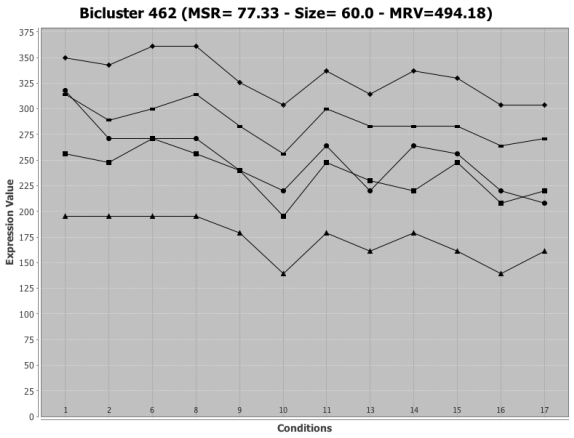

Bicluster 385 (MSR= 88.89 - Size= 165.0 - MRV=464.78)
Expression Value
Conditions

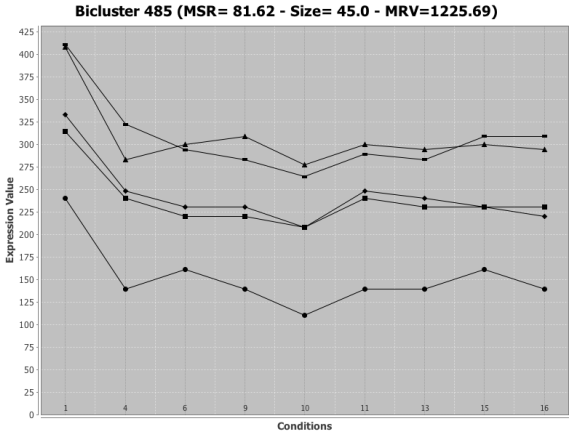

Bicluster 394 (MSR= 81.74 - Size= 60.0 - MRV=710.25)
Expression Value
Conditions

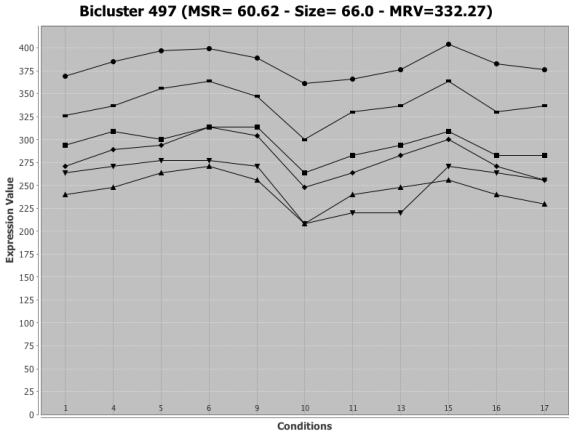

Bicluster 419 (MSR= 71.79 - Size= 65.0 - MRV=489.14)
Expression Value
Conditions

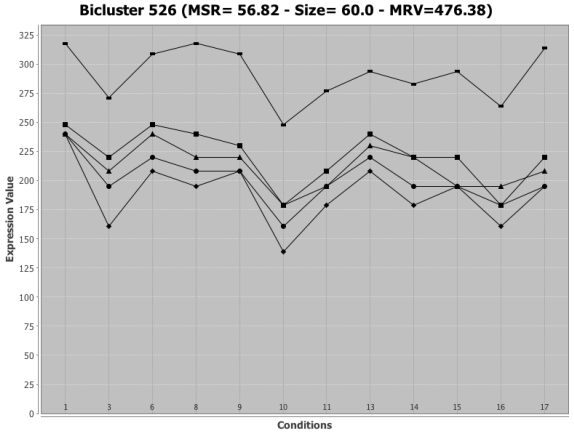

Bicluster 439 (MSR= 28.15 - Size= 65.0 - MRV=331.42)
Expression Value
Conditions

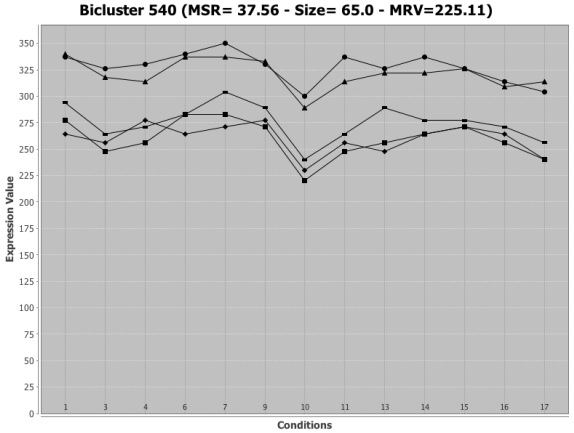

Bicluster 453 (MSR= 49.31 - Size= 60.0 - MRV=252.85)
Expression Value
Conditions

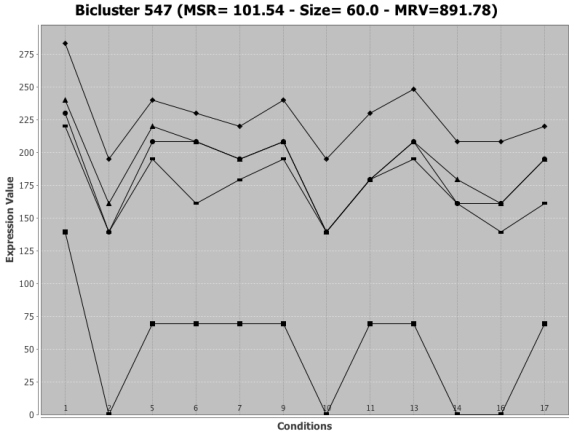

Bicluster 462 (MSR= 77.33 - Size= 60.0 - MRV=494.18)
Expression Value
Conditions

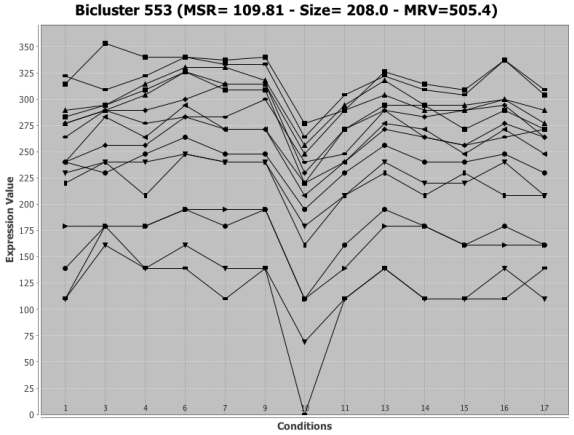

Bicluster 485 (MSR= 81.62 - Size= 45.0 - MRV=1225.69)
Expression Value
Conditions


Bicluster 497 (MSR= 60.62 - Size= 66.0 - MRV=332.27)
Expression Value
Conditions


Bicluster 526 (MSR= 56.82 - Size= 60.0 - MRV=476.38)
Expression Value
Conditions


Bicluster 540 (MSR= 37.56 - Size= 65.0 - MRV=225.11)
Expression Value
Conditions

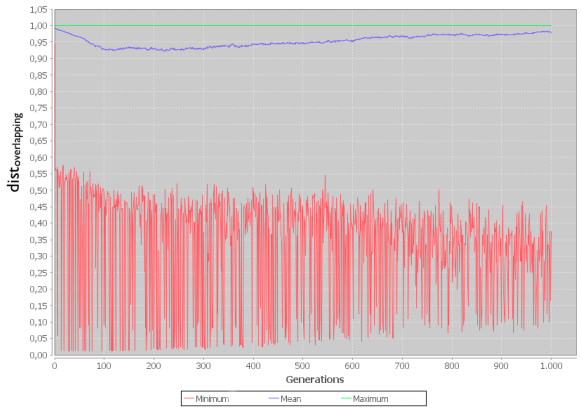

Bicluster 547 (MSR= 101.54 - Size= 60.0 - MRV=891.78)
Expression Value
Conditions


Bicluster 553 (MSR= 109.81 - Size= 208.0 - MRV=505.4)
Expression Value
Conditions

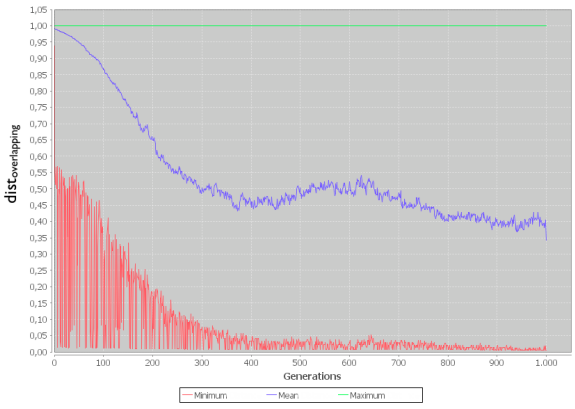
These biclusters exhibit clearly these "strikingly similar up-regulation and down-regulation" of some genes under some conditions put forward by Cheng and Church. Logically, these biclusters have very low MSR values (typically lower than 100) with satisfying values of MRV. This underlines the ability of MOBPEOC to find very coherent and interesting biclusters.
One should also notice the variety of the different patterns exhibited by these biclusters, which clearly correspond to different solutions to the biclustering problem. The mean overlapping between these biclusters is only 4.24 %, with many biclusters that do not overlap. However, the overlapping level can also reach up to 77.33% between bicluster 251 and bicluster 394. These two last biclusters could represent two parts of a same maximal bicluster, but also two highly overlapping but different biclusters. The MOBPEOC method is thus able to find totally disjoint biclusters as much as highly overlapping ones. One can suppose that the weakly or non-overlapping biclusters come from probabilistic individuals originating from different niches, while the highly overlapping ones come from probabilistic individuals originating from the same niche. It can also be remarked that the set of biclusters covers all the 17 conditions in the dataset. Finally, the remanence of some regular patterns across the whole set of biclusters, especially involving conditions 6, 9, 10, 11, 13 and 16 (which are present in all the biclusters), should be noticed, and will be discussed in the next sub-section.

4.2.3 Overlapping control efficiency

The mean overlapping between the 600 biclusters returned using the MOBPEOC approach over the Yeast dataset is 16.10%. This value seems to agree with the choice of a niching radius enforcing 15% of similarity between the niches established in the GA. We pushed forward the analysis of the efficiency of the overlapping control mechanisms in MOBPEOC by running once again the whole algorithmic process, but with a niching radius assuming value 0, i.e. which allows 100% of similarity between the niches established in the GA.
The following graphs show the evolution of the minimal (red), mean (blue) and maximal (green) overlapping distance $d_{overlapping}$ between the individuals of the GA for each of the 1000 generations. The graph on the left shows this evolution when the niching radius assumes value 0.85, and the graph on the right when it assumes value 0.



Niching radius = 0.85

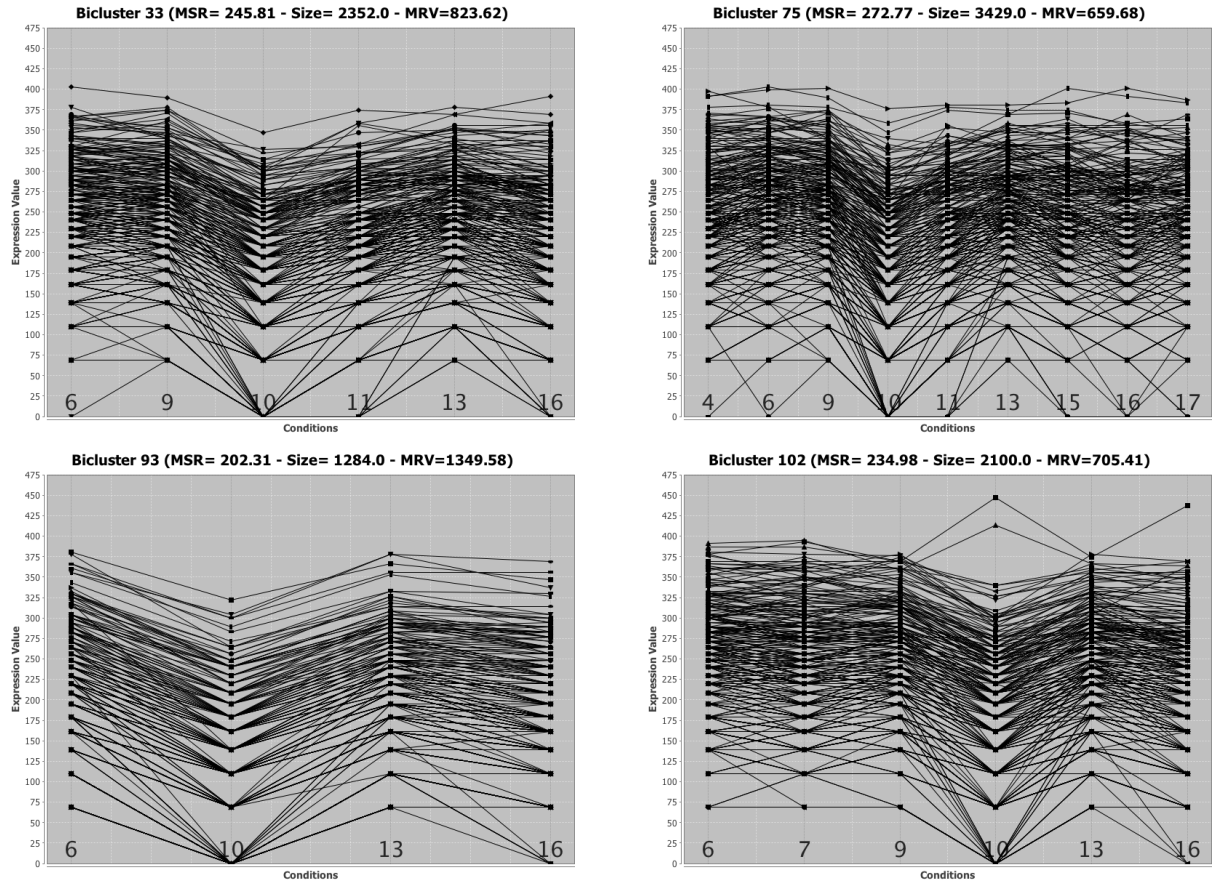


Niching radius = 0

When the niching radius assumes value 0.85, one can see that the mean overlapping distance between the individuals decreases during about 100 generations. We can suppose that this phase corresponds to the discovery of a core set of niches. In a second phase, the mean distance grows very slowly while the minimal overlapping distance continues to decrease. We can suppose that this phase corresponds to the convergence inside each niche towards a particularly good solution within the niche.
When the niching radius is zero, the mean and minimal overlapping distance collapse in about 400 generations. This difference of behavior between the algorithm with a zero niching radius and the one with a 0.85 niching radius is a clear hint of the efficiency of the overlapping control mechanism.

We have also compared the biclusters obtained using MOPEOC respectively with a 0.85 and a 0 niching radius in the GA. The following table presents compared statistics between the two sets of returned biclusters. It details the overlapping between the biclusters in each set, their size and the number of rows and columns they contain. Minimal, mean, maximal and standard deviation values are detailed. The percentage of rows, columns and elements in EM' these sets of biclusters cover is also presented. Below the table, the expression graph of the typical biclusters returned by MOBPEOC with a zero niching radius are presented, to be compared with the ones detailed in the previous subsection.

		Niching radius = 0.85	Niching radius = 0
Overlapping	Min	0.0%	8.17%
	Mean	16.10%	60.59%
	Max	100%	100%
	St. Dev.	22.67%	18.63%
Size	Min	30	1284
	Mean	391.34	3288.61
	Max	2805	4836
	St. Dev.	481.63	2178.46
# Genes	Min	5	261
	Mean	43.58	389.79
	Max	279	442
	St. Dev.	54.00	29.52
# Conditions	Min	5	4
	Mean	9.23	8.36
	Max	13	12
	St. Dev.	1.46	1.98
Coverage	Genes	56.83%	44.14%
	Conditions	100%	88.24%
	Matrix	39.39%	28.92%



When the niching radius is set to zero, the returned biclusters become logically very similar, showing very high levels of overlapping, and covering less elements in EM'. They also become much bigger, with a much larger number of rows but a smaller number of columns, and they do not cover anymore all the columns of EM'.

Visual inspection of their expression graphs show that these biclusters typically contain a lot of rows exhibiting some similar patterns for a limited number of columns, involving typically some or all the columns 6, 9, 10, 11, 13 and 16. These patterns are the ones that appear in a recurrent way in the biclusters discovered with a niching radius assuming value 0.85.

Comparing our results with the ones of Cheng and Church [Cheng and Church, 2000], it appears that these biclusters found with a zero niching radius seem to be linked to the biclusters 46, 54 and 90 discovered by Cheng and Church. These biclusters are presented by Cheng and Church as representing shared patterns between many clusters of rows, found using clustering algorithms.

Without niching pressure, the probabilistic GA is able to individuate, with a low level of doubt, a very limited group of columns (6, 9, 10, 11, 13 and 16) for the ones a large number of rows exhibit similar highly varying patterns, shared between many real biclusters present in the data. When a strong niching pressure is applied, but still allowing a limited similarity between the partial combinations of rows and columns found, biclusters involving highly disjoint sets of rows are discovered, which share these similar and highly varying patterns individuated by the zero niching radius GA.

4.2.4 Performance comparison with concurrent techniques

In this subsection, we compare the results obtained using MOBPEOC on the Yeast dataset with the ones returned by the Cheng and Church [Cheng and Church, 2000], FLOC [Yang et al., 2003], SEBI [Divina and Aguilar-Ruiz, 2006] and SMOB [Divina and Aguilar-Ruiz, 2007] biclustering technique (see chapter 2).
The following tables compare statistics over the results produced using these approaches. The MSR, $Size$ and MRV of the returned biclusters are detailed, with the number of rows and columns they contain, and their level of covering of the rows, columns and elements in EM'.

	MSR				**Size**			
	Min	**Mean**	**Max**	**St. Dev.**	**Min**	**Mean**	**Max**	**St. Dev.**
MOBPEOC	0.00	88.83	308.36	57.46	30	391.34	2805	481.63
SEBI	-	205.18	-	4.49	-	209.92	-	171.39
SMOB	-	206.17	-	15.82	-	453.48	-	231.76
C&C	-	204.29	-	42.78	-	1576.98	-	2178.46
FLOC	-	187.54	-	-	-	1825.78	-	-

	# Genes				**# Conditions**			
	Min	**Mean**	**Max**	**St. Dev.**	**Min**	**Mean**	**Max**	**St. Dev.**
MOBPEOC	5	43.58	279	54.00	5	9.23	13	1.46
SEBI	-	13.61	-	10.38	-	15.25	-	1.37
SMOB	-	27.28	-	14.88	-	15.46	-	1.88
C&C	-	166.71	-	226.37	-	12.09	-	4.39
FLOC	-	195	-	-	-	12.8	-	-

	MRV				**Coverage**		
	Min	**Mean**	**Max**	**St. Dev.**	**Genes**	**Conditions**	**Matrix**
MOBPEOC	0.00	410.27	2266.08	232.86	56.83 %	100 %	39.39 %
SEBI	-	-	-	-	43.55 %	100 %	38.14 %
SMOB	-	-	-	-	47.02 %	100 %	40.39 %
C&C	-	-	-	-	97.12 %	100 %	81.14 %

The MOPBEOC results seem very good compared to the ones returned in the SEBI/SMOB approach, as they present very lower MSRs without really penalizing the *size* and coverage of the discovered biclusters. The results returned by SEBI/SMOB seem to contain more columns but less rows than the ones returned by MOBPEOC. Visual inspection of the results returned by both approach confirm that MOBPEOC returns much more coherent biclusters than SEBI/SMOB.
Comparing the MOBPEOC results with the Cheng and Church's and FLOC algorithms ones is a much more difficult task. Statistics for the Cheng and Church's technique, extracted from [Divina and Aguilar-Ruiz, 2007], are distorted as they include huge but flat biclusters returned by the method. Moreover, both these techniques allow to take into account rows with inverted expression patterns inside the biclusters, which is still not a feature of MOBPEOC. FLOC also allows to deal with missing values in the dataset without replacing them by random values, which is still not possible in MOBPEOC and Cheng and Church's algorithm. Despite these remarks, a comparison involving the statistics over the MSR and *size* seems to show that MOBPEOC allow to find much more coherent

but much smaller biclusters. Nevertheless, this should also be nuanced, as MOBPEOC uses a decision making process that looks for low MSR biclusters around the less doubtful solution in an individuated region, while C&C and FLOC search for maximal δ-biclusters, with $\delta = 300$. As MSR and $size$ are antagonist criteria, using a smaller value of δ in C&C or FLOC should typically return more coherent but smaller biclusters, while searching for bigger biclusters in MOBPEOC should return less coherent ones.

4.3 Biclustering of the Human dataset

4.3.1 Configuration of the algorithms

We detail and comment here the parameter values that were used within MOBPEOC for the biclustering of the Human dataset.

A Configuration of the genetic algorithm

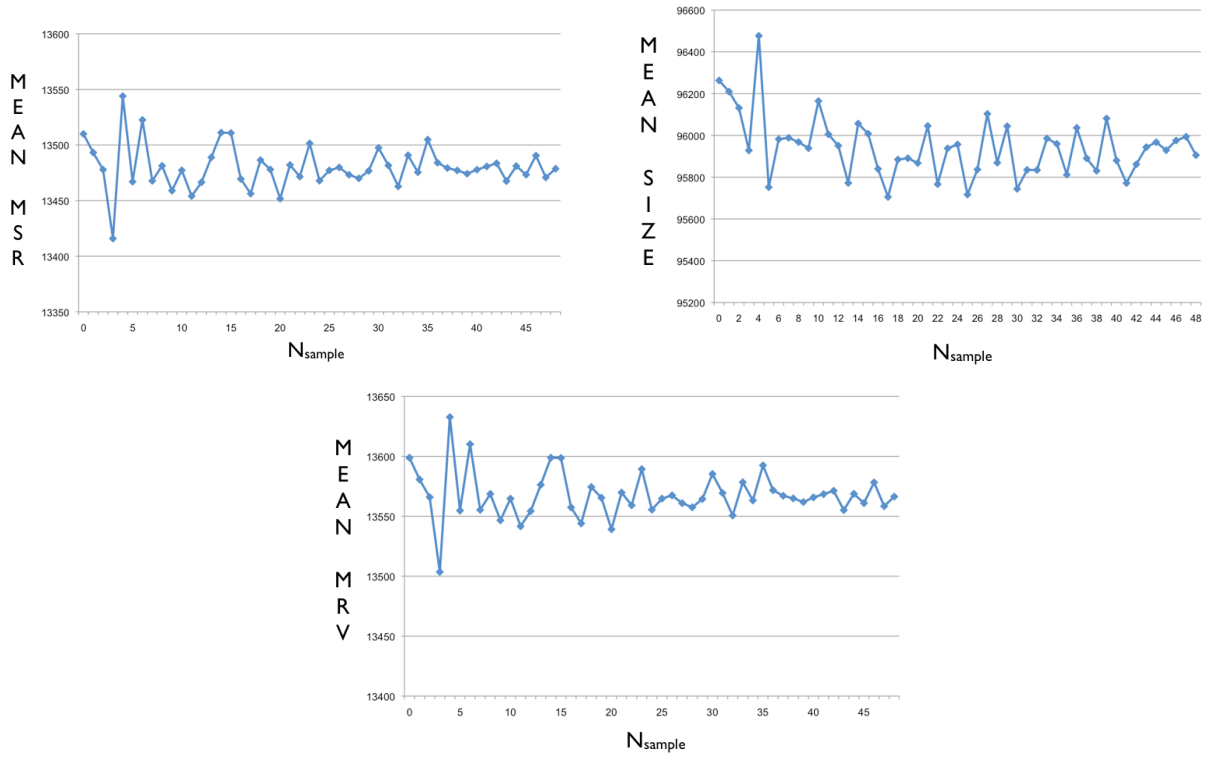
The parameters of the GA for the Human dataset, detailed in the figure below, are derived from the ones for the Yeast dataset.

```
// Generations loop (2)
maxNumberOfGenerations = 1000 ;
numberOfIndividualsPerGeneration = 600 ;

// Creation of new individuals (8)
thresholdUpRow = 0.7 ;
thresholdUpColumn = 0.7 ;
thresholdDownRow = 0.3 ;
thresholdDownColumn = 0.3 ;
minProportionOfSelectedRowsAtInitialization = 0.0005 ;
maxProportionOfSelectedRowsAtInitialization = 0.02 ;
minProportionOfSelectedColumnsAtInitialization = 0.90 ;
maxProportionOfSelectedColumnsAtInitialization = 1.0 ;

// Reinitialization process (2)
numberOfReinitializations = 30 ;
maxCyclesOfReinitialization = 990 ;

// Selection operator (4)
sizeOfTheEvaluationSampleSets = 50 ;
sizeOfComparisonSet = 510 ;
nichingRadius = 0.85 ;
scalingFactor = 1 ;

// Reproduction operators (3)
reproductionProbabilityRate = 0.85 ;
rowMeanCrossoverRate = 0.04 ;
columnMeanCrossoverRate =  0.01 ;

// Mutation operators (2)
mutationProbabilityRate = 0.05 ;
rowMutationRate = 0.6 ;
```

The algorithm was first run over the Human dataset using the parameters for the Yeast as a starting point. Then, these parameters were tuned to find different interesting partial combinations of rows and columns. The differences between the Human and the Yeast configurations are small, which is a hint of the stability of the method:

- The proportions values for the creation of new individuals have been adapted for the Human dataset. The goal was to allow the sub-matrices in the regions defined by these individuals to contain a number of rows an columns of the order of the ones of the biclusters discovered using the Cheng and Church and SEBI/SMOB approaches.
- Due to the larger number of rows and columns in the Human dataset, compared to the Yeast dataset, the row mean crossover operator is applied more frequently, while a column mean crossover operator is introduced, in order to enforce a proper exploration of the set of possible combinations of rows and columns.

As for the Yeast dataset, we computed the objective function values for a population of 200 random probabilistic individuals with different sample set sizes. For each sample set size N_{sample}, the mean of each of the objective function values for the 200 individuals is reported in the following graphs:



A size of 50 for the sample sets, as for Yeast dataset, seems sufficient for an effective evaluation of the probabilistic individuals.

B Configuration of the decision-making process

```
// Decision making process (4)
pMinRow = 0.15 ;
pMaxRow = 0.98 ;
pMinColumn = 0.15 ;
pMaxColumn = 0.95 ;
```

The probability values were tuned to promote a quick and efficient discovery of interesting low MSR biclusters, close to the less doubtful solution of the explored region.

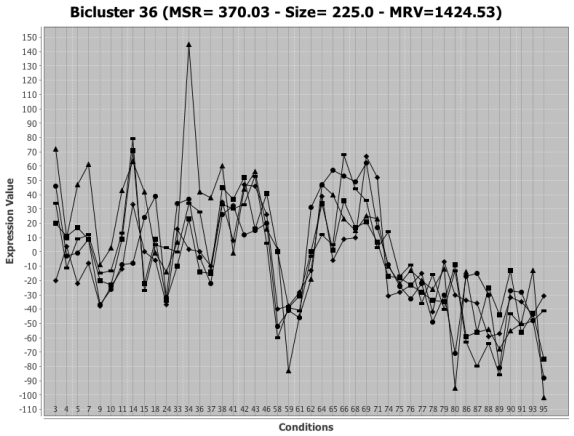
C Configuration of the simulated annealing algorithm

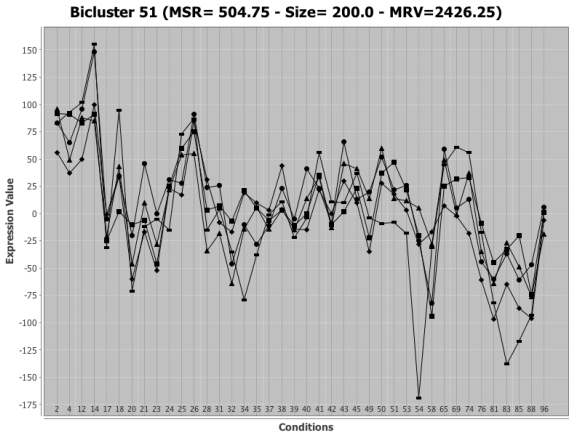
```
// Temperature (4)
initialRowTemperature = 2.0 ;
initialColumnTemperature = 30 ;
finalRowTemperature = 0.0375 ;
finalColumnTemperature = 0.5625 ;

// Other parameters (2)
numberOfAttempsPerTemperature = number of doubtful rows
                                + number of doubtful columns ;
temperatureDecreaseRatio = 0.85 ;
```

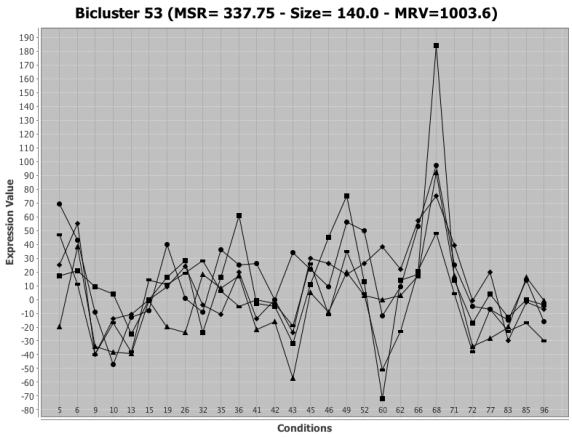
These values have been chosen using the rules detailed in section 3.5.4.

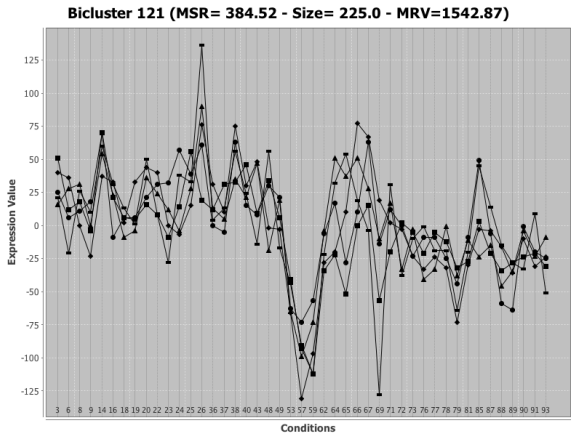
4.3.2 Quality of the discovered biclusters

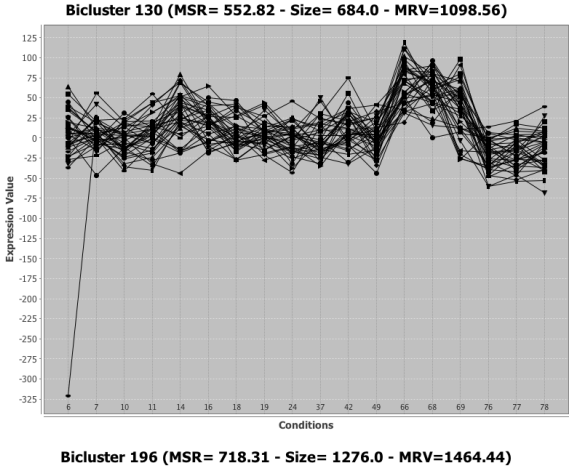
In this subsection, we show and comment the expression graphs of a panel of biclusters extracted from the 600 biclusters returned by the execution of MOBPEOC over the Human dataset. The rows and columns that compose the biclusters exposed here are detailed in appendix B.

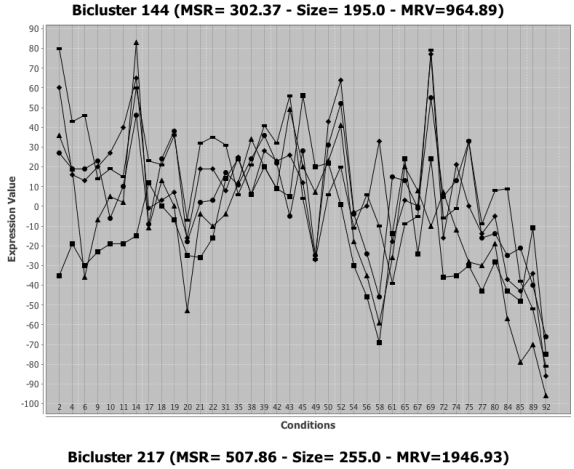


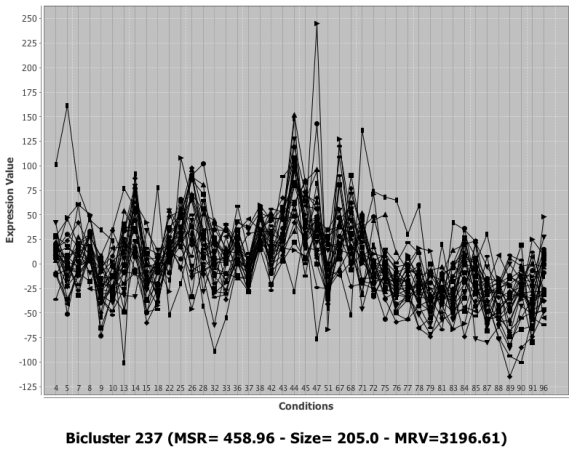


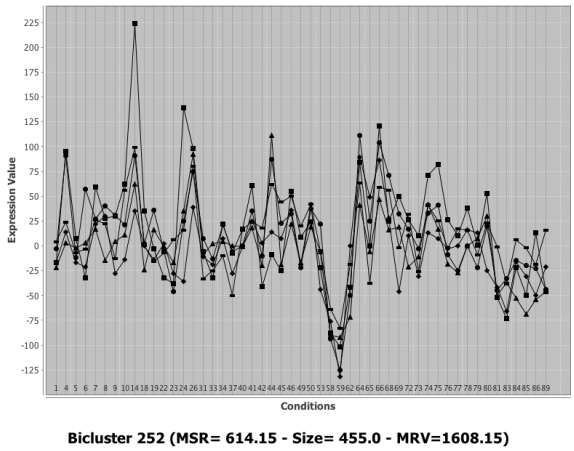

Bicluster 53 (MSR= 337.75 - Size= 140.0 - MRV=1003.6)
Expression Value
Conditions

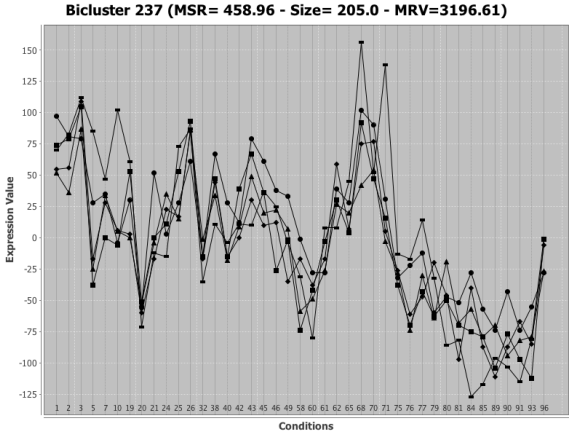

Bicluster 121 (MSR= 384.52 - Size= 225.0 - MRV=1542.87)
Expression Value
Conditions

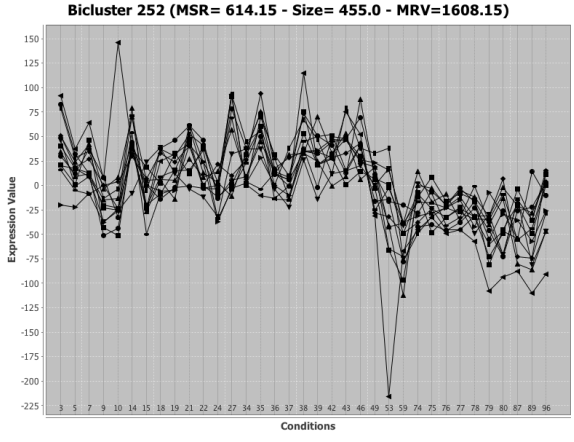

Bicluster 130 (MSR= 552.82 - Size= 684.0 - MRV=1098.56)
Expression Value
Conditions

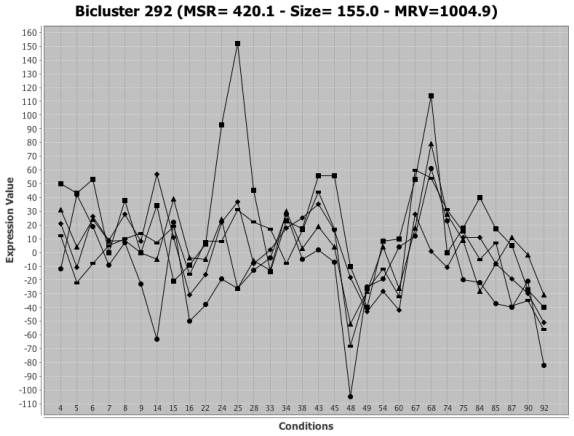

Bicluster 144 (MSR= 302.37 - Size= 195.0 - MRV=964.89)
Expression Value
Conditions

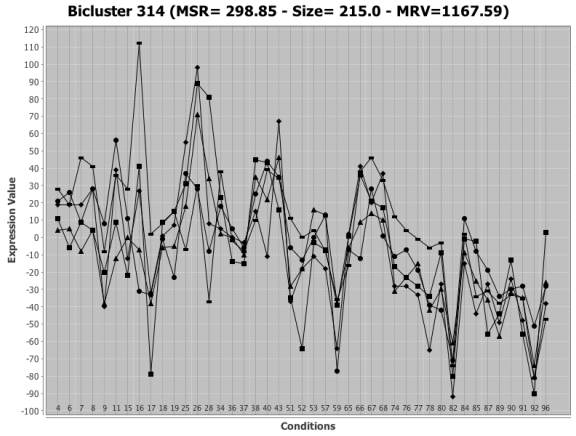

Bicluster 196 (MSR= 718.31 - Size= 1276.0 - MRV=1464.44)
Expression Value
Conditions

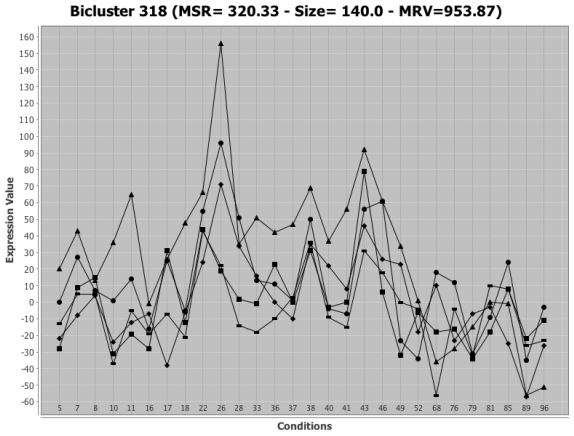

Bicluster 217 (MSR= 507.86 - Size= 255.0 - MRV=1946.93)
Expression Value
Conditions

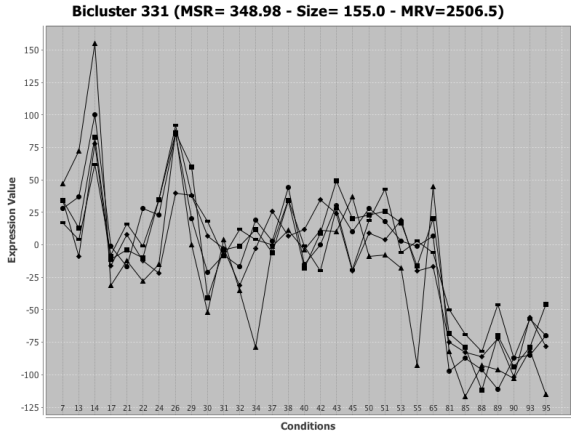

Bicluster 237 (MSR= 458.96 - Size= 205.0 - MRV=3196.61)
Expression Value
Conditions

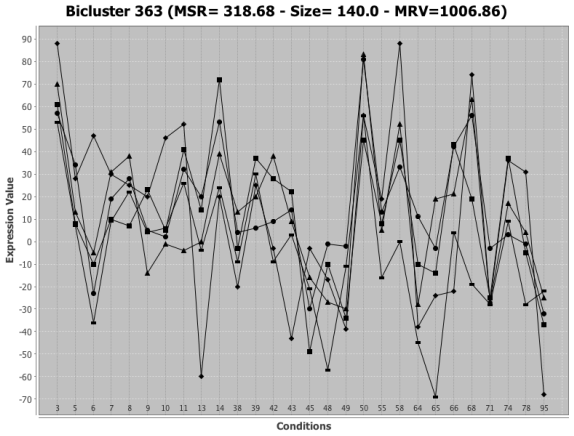

Bicluster 252 (MSR= 614.15 - Size= 455.0 - MRV=1608.15)
Expression Value
Conditions

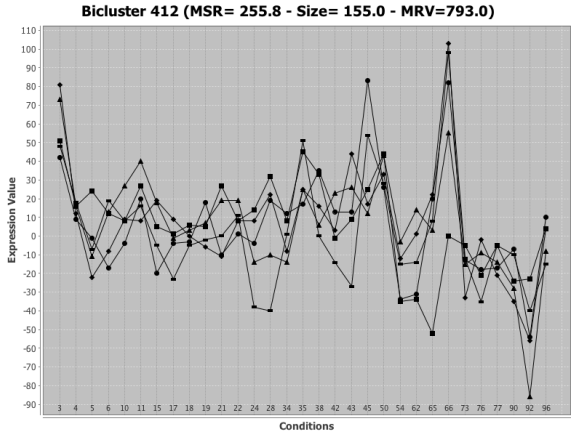

Bicluster 292 (MSR= 420.1 - Size= 155.0 - MRV=1004.9)
Expression Value
Conditions

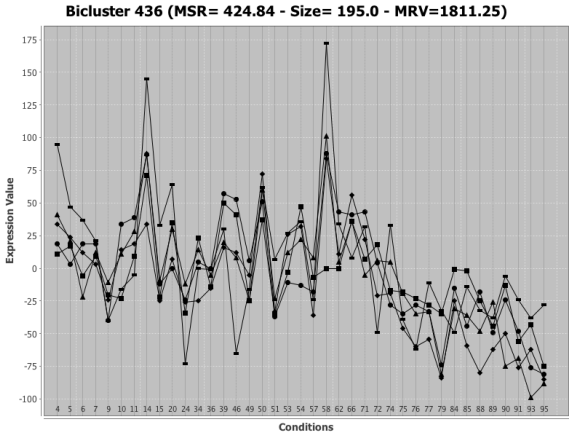

Bicluster 314 (MSR= 298.85 - Size= 215.0 - MRV=1167.59)
Expression Value
Conditions

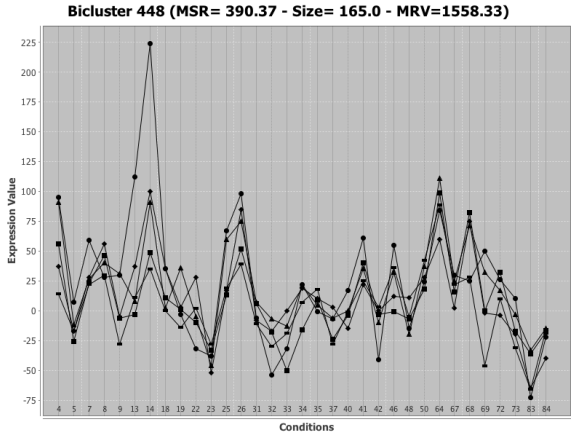

Bicluster 318 (MSR= 320.33 - Size= 140.0 - MRV=953.87)
Expression Value
Conditions

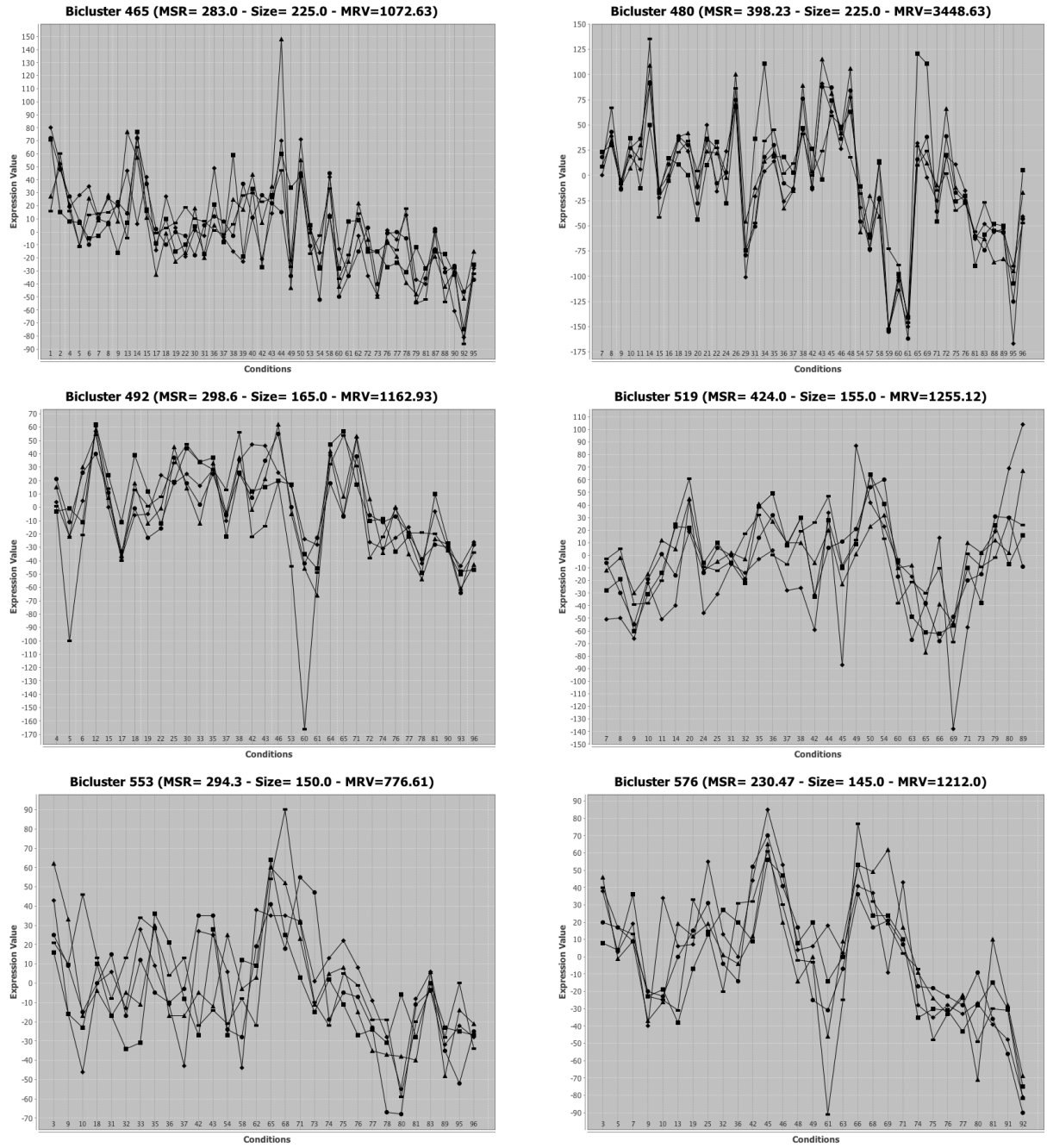

Bicluster 331 (MSR= 348.98 - Size= 155.0 - MRV=2506.5)
Expression Value
Conditions


Bicluster 363 (MSR= 318.68 - Size= 140.0 - MRV=1006.86)
Expression Value
Conditions


Bicluster 412 (MSR= 255.8 - Size= 155.0 - MRV=793.0)
Expression Value
Conditions


Bicluster 436 (MSR= 424.84 - Size= 195.0 - MRV=1811.25)
Expression Value
Conditions


Bicluster 448 (MSR= 390.37 - Size= 165.0 - MRV=1558.33)
Expression Value
Conditions



The tendencies observed over the Yeast dataset were confirmed with the Human dataset, which is a good hint of the stability of the method.

The biclusters discovered exhibit clearly the "strikingly similar up-regulation and down-regulation" of some genes under some conditions put forward by Cheng and Church. They have low MSR values (lower than 800 and even often lower than 600) and good MRV values.

They also exhibit many different patterns, and with only 3.16% of mean overlapping, where most biclusters simply do not overlap, they clearly represent different solutions to the biclustering problem. The recurrence of some patterns can nevertheless be observed in several biclusters, and the level of overlapping can reach up to 49.09 between biclusters 217 and bicluster 448. The whole set of 600 biclusters has a mean overlapping of 10.71%, which seems to agree with the choice of a niching radius enforcing 15% of similarity between the niches established in the GA.

4.3.3 Performance comparison with concurrent techniques

As with the Yeast dataset, we compare here the results obtained using MOBPEOC on the Human dataset with the ones returned by the Cheng and Church, SEBI and SMOB biclustering techniques (no data were found for the FLOC algorithm with the Human dataset).

	MSR				**Size**			
	Min	**Mean**	**Max**	**St. Dev.**	**Min**	**Mean**	**Max**	**St. Dev.**
MOBPEOC	159.64	744.04	1629.42	337.21	70	1967.05	9000	2000.54
SEBI	-	1028.84	-	29.19	-	615.84	-	278.35
SMOB	-	1019.16	-	120.78	-	709.13	-	378.05
C&C	-	850.04	-	153.91	-	4595.98	-	3353.72

	# Genes				**# Conditions**			
	Min	**Mean**	**Max**	**St. Dev.**	**Min**	**Mean**	**Max**	**St. Dev.**
MOBPEOC	5	54.34	217	53.51	14	38.03	68	9.99
SEBI	-	14.07	-	5.39	-	43.57	-	6.20
SMOB	-	11.60	-	12.55	-	78.47	-	19.46
C&C	-	269.22	-	204.71	-	24.5	-	20.92

	MRV				**Coverage**		
	Min	**Mean**	**Max**	**St. Dev.**	**Genes**	**Conditions**	**Matrix**
MOBPEOC	448.10	1186.33	3448.64	331.67	33.18 %	100 %	21.92 %
SEBI	-	-	-	-	38.23 %	100 %	34.07 %
SMOB	-	-	-	-	45.05 %	100 %	33.52 %
C&C	-	-	-	-	91.58 %	100 %	36.81 %

Here again, the tendencies observed over the Yeast dataset were confirmed with the Human dataset.
The results returned by MOBPEOC have a much lower mean MSR than the ones returned by SEBI/SMOB, without impacting the *size* of the discovered biclusters. The biclusters obtained using SEBI/SMOB seem to exhibit a better coverage of the expression matrix EM'. This tendency does not exist with the Yeast dataset. This is probably linked to the sequential coverage technique used in SEBI/SMOB, which prevents overlapping between the individuated biclusters and thus enforces a better coverage, coupled with the increased size of EM' for the Human dataset compared to the Yeast dataset.
Comparisons between MOBPEOC and the Cheng and Church's algorithm are difficult for the same reasons that the ones detailed for the Yeast dataset. Here again, MOBPEOC returns results with a lower MSR but also a lower *size* than those returned by Cheng and Church's algorithm with $\delta = 1000$. It should be noticed that many biclusters presented by Cheng and Church over the Human dataset contain a lot of rows with inverted patterns. Encompassing the search for such rows within MOBPEOC could thus improve in an important way the results (and typically, their *size*) returned by MOBPEOC over the Human dataset.

Part IV

Conclusions and future work

Conclusions and future work

Biclustering is a data mining problem that consists in finding sub-matrices of a given matrix that optimize several quality criteria, enforcing essentially a coherence requirement. This problem has many interesting applications in many different domains. In the biological context of gene expression data analysis, biclustering techniques can allow to find particular groups of genes exhibiting a similar expression pattern under a particular group of conditions. This is of particular importance from a biological (like for gene profiling) and medical (like for a better understanding of diseases) point of view.
The first work over this biological instance of the biclustering problem was proposed by Cheng and Church in 2000. They defined a mathematical specification of the problem, known as the δ-bicluster model, as well as a greedy algorithmic approach to solve it.
As biclustering of gene expression data is a hard multimodal and multi-objective combinatorial optimization problem, the search for interesting solutions can benefit from the use of evolutionary computation. Evolutionary computation is a non strictly delimited class of algorithms, sharing the use of mechanisms inspired by darwinian evolution to solve optimization problems. Evolutionary computation was notably used by F. Divina to solve the gene expression data biclustering problem, according to the δ-bicluster model specification. This approach lead to the discovery of interesting biclusters by the SEBI and SMOB genetic algorithms.

In this thesis, we developed a Multi-Objective evolutionary Biclustering genetic algorithm with Probabilistic Encoding and Overlapping Control (MOBPEOC) to solve the gene expression data biclustering problem. Based on the δ-bicluster model specification and inspired by the SEBI/SMOB approach, MOBPEOC introduces new mechanisms to offer an improved biclustering process:

- The use of a multi-objective biclustering approach is necessary, as the individuated biclusters should be at the same time very coherent and maximal, and they should also show high variations of the expression levels of their genes under their different conditions. Nevertheless such a multi-objective approach is also highly problematic for individuating particular biclusters, as these objectives are particularly antagonist. The adequacy of a particular gene or condition in a bicluster can thus not always be established formally in a precise way.

  MOPBEOC allows to deal with this problem in an elegant and totally new way. The search phase of the multi-objective solving process is enforced by a GA using probabilistic encoding, a new evolutionary technique that we propose and introduce within MOBPEOC. This technique allows to express a given level of doubt over the adequacy of a row or a column in the solution. Probabilistic encoding also increases the exploration power of the GA. One can then exploit a-posteriori the

results returned by this probabilistic GA to individuate precise biclusters according to a chosen decision-making policy.

In this thesis, a simulated annealing local search method exploiting the results of the GA is proposed to show the ability of the method to individuate highly coherent biclusters. These individuated biclusters could potentially be used as a starting point for a human-driven decision-making process, where a computer-assisted biologist would add and remove rows and columns to the candidate bicluster, to establish a biologically significant maximal one.

- Biclustering of gene expression data is a multimodal problem, as different biclusters hinting biological relations between different sets of genes and conditions should be individuated in the data. Such different biclusters should not share too much elements in the expression matrix, but can nevertheless exhibit some level of overlapping.

  In the MOBPEOC probabilistic GA, we combine in a new way two existing evolutionary techniques, sharing and niched Pareto selection, in order to individuate several different partial combinations of genes and conditions, which all optimize simultaneously coherence, size and variance of the genes expression levels. This technique introduces a parameter, the niching radius, that allows to specify the level of overlapping between the different individuated combinations of rows and conditions.

An important part of the work achieved during this thesis consisted in testing MOBPEOC over two real gene expression datasets, typically used in the literature to test the many proposed biclustering techniques: the Yeast dataset and the Human dataset. The obtained results confirm the efficiency and the stability of the method.
For both datasets, biclusters showing "strikingly similar up-regulation and down-regulation" of some genes under some conditions are discovered. These biclusters have a very low mean squared residue, much lower than in the biclusters returned by the SEBI/SMOB approach, combined with a sufficient mean row variance, and without impacting the size of the biclusters, compared to the SEBI/SMOB results.
The results also demonstrate the ability of MOBPEOC to find different biclusters and to control the mean level of overlapping between them. The individuated biclusters present many different variation patterns, and can cover from totally disjoint to highly overlapping sets of elements in the expression matrix.
These particularly interesting results are a strong hint of the efficiency of the new evolutionary mechanisms developed in the MOBPEOC GA. This GA was notably a first conclusive life-size test for the probabilistic encoding technique, which could be particularly useful to deal with uncertainty in the frame of many other problems to solve using evolutionary computation.

The comparison of the MOBPEOC results with the ones returned by classical biclustering techniques, like Cheng and Church's algorithm is inconclusive and reveals the limits of the approach used in this thesis. The work presented here is indeed limited (for time reasons) to an improvement of the evolutionary biclustering approach proposed by the SEBI/SMOB algorithms, in the restricted frame of the δ-bicluster model.
Future work on the MOBPEOC approach should evaluate the adequacy of the δ-bicluster model, and be based on a wider review of the original needs of biologists searching for

biclusters in gene expression data and of the other bicluster models existing in the literature.
Moreover, the actual MOBPEOC approach can be improved in several ways:

- The approach should deal explicitly with missing data in the studied dataset, instead of replacing them by random values, and should take care of the possible measurement variations within the data, through a discussed normalization procedure.
- The approach should be able to deal with biclusters containing rows whose expression trends are "mirror images" of each other. A possible idea to test would consist in using a probabilistic encoding with negative values, a negative value for a particular row indicating the additive inverse of the probability for the mirror image of the row to be coherent within the combination.
- The approach should use a decision making process tailored and tested for the practical requirements of the biologists who need to analyze gene expression data.

The validation of the obtained results and of the efficiency of the method can also be deepened.
The biological relevance of the obtained results could notably be evaluated by comparing them with the relations between genes and conditions already acknowledged by biologists in the Yeast and Human datasets. The method could also be applied to other existing gene expression datasets.
The comparison of the MOBPEOC results with the ones of the other methods could also be improved. This would require notably advanced tests using the other methods. A comparison of the adequacy of the overlapping control mechanisms between the several methods could also be very interesting.
The MOBPEOC approach combines several complex techniques (GA with probabilistic encoding, sharing, niched Pareto selection, reinitialization, simulated annealing) in a totally untested new way, and can be configured using a very large number of parameters. Despite the good results it produces, the method can thus be criticized for a complexity that prevents a trivial understanding of the dynamics of its search process. A more detailed theoretical and experimental analysis of the efficiency of each of the components of the method, and of the effect of combining them in a single algorithm, would be of particular interest to validate and improve the efficiency of the technique.

Part V

Bibliography

Bibliography

[Agrawal et al., 1998] Agrawal, R., Gehrke, J., Gunopulos, D. and Raghavan, P. (1998). Automatic subspace clustering of high dimensional data for data mining applications. SIGMOD Rec. *27*, 94–105. 49

[Alizadeh et al., 2000] Alizadeh, A. A., Eisen, M. B., Davis, R. E., Ma, C., Lossos, I. S., Rosenwald, A., Boldrick, J. C., Sabet, H., Tran, T., Yu, X., Powell, J. I., Yang, L., Marti, G. E., Moore, T., Hudson, J., Lu, L., Lewis, D. B., Tibshirani, R., Sherlock, G., Chan, W. C., Greiner, T. C., Weisenburger, D. D., Armitage, J. O., Warnke, R., Levy, R., Wilson, W., Grever, M. R., Byrd, J. C., Botstein, D., Brown, P. O. and Staudt, L. M. (2000). Distinct types of diffuse large B-cell lymphoma identified by gene expression profiling. Nature *403*, 503–511. 61

[Amant, 2010] Amant, S. (2010). Memetic algorithm for discovering biclusters in microarray data. Master's thesis University of Namur (FUNDP) - Computer Science Department. 70

[Bäck et al., 2000] Bäck, T., Fogel, D. and Michalewicz, Z. (2000). Evolutionary computation 2: advanced algorithms and operators, vol. 1,. IOP Publishing Ltd. 12

[Back et al., 1999] Back, T., Fogel, D. B. and Michalewicz, Z., eds (1999). Evolutionary Computation 1: Basic Algorithms and Operators. IOP Publishing Ltd., Bristol, UK, UK. 10, 11, 12

[Barichard, 2003] Barichard, V. (2003). Approches hybrides pour les problèmes multiobjectifs. PhD thesis, Ecole Doctorale d'Angers France. 13

[Berkhin and Becher, 2002] Berkhin, P. and Becher, J. D. (2002). Learning simple relations: Theory and applications. In In Second SIAM Data Mining Conference pp. 420–436, SIAM. 49

[Berrer et al., 2003] Berrer, D., Dubitzky, W. and Draghici, S. (2003). A practical approach to microarray data analysis chapter 1. .: Kluwer Academic Publishers. 46, 48, 53

[Beyer, 2001] Beyer, H.-G. (2001). The theory of Evolution Strategies, Natural Computing Series. Springer. 10

[Beyer et al., 2002] Beyer, H.-G., De Jong, K., Reeves, C. and Wegener, I., eds (2002). Dynamics of Evolutionary Algorithms on Infinite Search Spaces (J. E. Rowe) in Theory of Evolutionary Algorithms Dagstuhl, Germany. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany. 14

[Bleuler et al., 2004] Bleuler, S., Prelic, A. and Zitzler, E. (2004). An EA Framework for Biclustering of Gene Expression Data. In Proceedings of the 2004 congress on evolutionary computation, (IEEE, ed.), pp. 166–173, IEEE. 69, 70

[Bryan, 2005] Bryan, K. (2005). Biclustering of Expression Data Using Simulated Annealing. In CBMS '05: Proceedings of the 18th IEEE Symposium on Computer-Based Medical Systems pp. 383–388, IEEE Computer Society, Washington, DC, USA. 46, 62, 92

[Bulcke, 2007] Bulcke, T. V. D. (2007). ProBic: identification of overlapping biclusters using Probabilistic Relational Models, applied to simulated gene expression data. `http://videolectures.net/pmnp07_bulcke_piof/`. 61

[Burjorjee, 2009] Burjorjee, K. M. (2009). Generative Fixation A Unified Explanation for the Adaptive Capacity of Simple Recombinative Genetic Algorithms. PhD thesis, Computer Science Department, Brandeis University. 29

[Chelouah and Siarry, 2000] Chelouah, R. and Siarry, P. (2000). A continuous Genetic Algorithm Designed for the Global Optimization. Journal of Heuristics *6*, 191–213. 14

[Cheng and Church, 2000] Cheng, Y. and Church, G. M. (2000). Biclustering of Expression Data. In Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology pp. 93–103, AAAI Press. 47, 49, 50, 58, 60, 69, 108, 109

[Cho et al., 1998] Cho, R. J., Campbell, M. J., Winzeler, E. A., Steinmetz, L., Conway, A., Wodicka, L., Wolfsberg, T. G., Gabrielian, A. E., Landsman, D., Lockhart, D. J. and Davis, R. W. (1998). A genome-wide transcriptional analysis of the mitotic cell cycle. Mol Cell *2*, 65–73. 61

[Deb et al., 2000] Deb, K., Pratap, A., Agarwal, S. and Meyarivan, T. (2000). A Fast Elitist Multi-Objective Genetic Algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation *6*, 182–197. 42

[Dhillon, 2001] Dhillon, I. S. (2001). Co-clustering documents and words using bipartite spectral graph partitioning. In KDD '01: Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining pp. 269–274, ACM, New York, NY, USA. 49

[Divina, 2008] Divina, F. (2008). Probabilistic encoding: a novel representation for evolutionary algorithms. (Unpublished). 75, 77, 78

[Divina and Aguilar-Ruiz, 2006] Divina, F. and Aguilar-Ruiz, J. S. (2006). Biclustering of Expression Data with Evolutionary Computation. IEEE Transactions on Knowledge and Data Engineering *18*, 590–602. 47, 62, 69, 109

[Divina and Aguilar-Ruiz, 2007] Divina, F. and Aguilar-Ruiz, J. S. (2007). A Multi-Objective Approach to Discover Biclusters in Microarray Data. In Proceedings of the 16th Genetic and Evolutionary Computation Conference (GECCO-2007) p. to appear, ACM. 47, 62, 68, 69, 109

[Dréo et al., 2003] Dréo, J., Pétrowski, A., Siarry, P. and Taillard, E. (2003). Métaheuristiques pour l'optimisation difficile. Eyrolles, Paris. 10, 12, 30, 42, 93, 94, 95, 102, 103

[Eiben and Smith, 2003] Eiben, A. and Smith, J. (2003). Introduction to evolutionary computing. Springer. 12, 31, 32

[Fei and Juan, 2008] Fei, L. and Juan, L. (2008). Biclustering of Gene Expression Data with a New Hybrid Multi-Objective Evolutionary Algorithm of NSGA-II and EDA. In Bioinformatics and Biomedical Engineering, 2008. ICBBE 2008. The 2nd International Conference on pp. 1912 – 1915, IEEE. 70

[Fogel et al., 1966] Fogel, L., Owens, A. and Walsh, M. (1966). Artificial Intelligence through Simulated Evolution. Wiley -. 10

[Fogel, 1999] Fogel, L. J. (1999). Intelligence through simulated evolution: forty years of evolutionary programming. John Wiley and Sons, Inc., New York, NY, USA. 10

[Gallo et al., 2009] Gallo, C. A., Carballido, J. A. and Ponzoni, I. (2009). Microarray Biclustering: A Novel Memetic Approach Based on the PISA Platform. In EvoBIO '09: Proceedings of the 7th European Conference on Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics pp. 44–55, Springer-Verlag, Berlin, Heidelberg. 69, 70, 75

[Garey and Johnson, 1990] Garey, M. R. and Johnson, D. S. (1990). Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York, NY, USA. 16

[Goldberg, 1989] Goldberg, D. E. (1989). Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. 10, 12, 16, 41

[Goldberg et al., 1992] Goldberg, D. E., Deb, K. and Horn, J. (1992). Massive Multimodality, Deception, and Genetic Algorithms. 38

[Goldberg and Richardson, 1987] Goldberg, D. E. and Richardson, J. (1987). Genetic algorithms with sharing for multimodal function optimization. In Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application pp. 41–49, L. Erlbaum Associates Inc., Hillsdale, NJ, USA. 32

[Hancock, 1992] Hancock, P. J. B. (1992). Coding strategies for genetic algorithms and neural nets. PhD thesis, Department of Computer Science, University of Stirling. 26

[Hao et al., 1999] Hao, J.-K., Galinier, P. and Habib, M. (1999). Metaheuristiques pour l'optimisation combinatoire et l'affectation sous contraintes. Revue d'Intelligence Artificielle (Hermes) *13*, 283–324. 14

[Hartigan, 1972] Hartigan, J. A. (1972). Direct Clustering of a Data Matrix. Journal of the American Statistical Association *67*, 123–129. 49

[Hofmann and Puzicha, 1999] Hofmann, T. and Puzicha, J. (1999). Latent Class Models for Collaborative Filtering. In IJCAI '99: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence pp. 688–693, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. 49

[Holland, 1975] Holland, J. H. (1975). Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor, MI, USA. 10

[Horn et al., 1994] Horn, J., Nafpliotis, N. and Goldberg, D. E. (1994). A niched Pareto genetic algorithm for multiobjective optimization. In Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on pp. 82–87 vol.1, IEEE. 41, 42, 79, 102

[Kirkpatrick et al., 1983] Kirkpatrick, S., Gelatt, C. D., J. and Vecchi, M. P. (1983). Optimization by Simulated Annealing. Science *220*, 671–680. 93

[Krasnogor and Gustafson, 2002] Krasnogor, N. and Gustafson, S. (2002). Toward Truly "Memetic" Memetic Algorithms: discussion and proofs of concept. In Advances in Nature-Inspired Computation: The PPSN VII Workshops. vol. 16(52), PEDAL (Parallel, Emergent and Distributed Architectures Lab). 30

[Lazzeroni and Owen, 2000] Lazzeroni, L. and Owen, A. (2000). Plaid Models for Gene Expression Data. Statistica Sinica *12*, 61–86. 50

[Liu and Motoda, 1998] Liu, H. and Motoda, H. (1998). Feature Selection for Knowledge Discovery and Data Mining. Kluwer Academic Publishers, Norwell, MA, USA. 78

[Lozano et al., 2006] Lozano, J. A., n. Larra, P., n. I., I. and Bengoetxea, E. (2006). Towards a New Evolutionary Computation: Advances on Estimation of Distribution Algorithms (Studies in Fuzziness and Soft Computing). Springer-Verlag New York, Inc., Secaucus, NJ, USA. 78

[Madeira and Oliveira, 2004] Madeira, S. C. and Oliveira, A. L. (2004). Biclustering algorithms for biological data analysis: a survey. IEEE/ACM Transactions on Computational Biology and Bioinformatics *1*, 24–45. 46, 47, 48, 49, 50, 59, 60, 61, 62

[Mitchell, 1996] Mitchell, M. (1996). An introduction to genetic algorithms. MIT Press, Cambridge, MA, USA. 43

[Mitra and Banka, 2006] Mitra, S. and Banka, H. (2006). Multi-objective evolutionary biclustering of gene expression data. Pattern Recogn. *39*, 2464–2477. 70

[NCBI, 2010] NCBI (2010). Microarrays: chipping away at the mysteries of science and medicine. `http://www.ncbi.nlm.nih.gov/About/primer/microarrays.html`. 46

[Nepomuceno et al., 2010] Nepomuceno, J. A., Troncos, A. and Aguilar-Ruiz, J. S. (2010). Evolutionary metaheuristic for biclustering based on linear correlations among genes. In SAC '10: Proceedings of the 2010 ACM Symposium on Applied Computing pp. 1143–1147, ACM, New York, NY, USA. 70

[Neumaier, 2010a] Neumaier, A. (2010a). Global Optimization. `www.mat.univie.ac.at/~neum/glopt.html`. 15

[Neumaier, 2010b] Neumaier, A. (2010b). Examples and Case Studies in Optimization. `www.mat.univie.ac.at/~neum/glopt/applications.html`. 15

[Oei et al., 1991] Oei, C., Goldberg, D. and Chang, S. (1991). Tournament Selection, Niching, and the Preservation of Diversity. Technical Report 01011 Illinois Genetic Algorithms Laboratory, University of Illinois Urbana-Champaign. 36, 37

[Paenke et al., 2006] Paenke, I., Branke, J. and Jin., Y. (2006). Efficient search for robust solutions by means of evolutionary algorithms and fitness approximation. IEEE Trans. Evolutionary Computation *10*, 405–420. 78

[Paz, 1997] Paz, C. E. (1997). A Survey of Parallel Genetic Algorithms. 23

[Rahnamayan et al., 2007] Rahnamayan, S., Tizhoosh, H. R. and Salama, M. M. A. (2007). A novel population initialization method for accelerating evolutionary algorithms. Comput. Math. Appl. *53*, 1605–1614. 18

[Rechenberg, 1965] Rechenberg, I. (1965). Cybernetic solution path of an experimental problem. Technical report Royal Air Force Establishment. 10

[Sareni and Krahenbuhl, 1998] Sareni, B. and Krahenbuhl, L. (1998). Fitness sharing and niching methods revisited. Evolutionary Computation, IEEE Transactions on *2*, 97–106. 38

[Singh and Deb, 2006] Singh, G. and Deb, Dr., K. (2006). Comparison of multi-modal optimization algorithms based on evolutionary algorithms. In GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation pp. 1305–1312, ACM, New York, NY, USA. 32, 33

[Stender, 1993] Stender, J. (1993). Parallel Genetic Algorithms: Theory and Applications. IOS Press, Amsterdam, The Netherlands, The Netherlands. 23

[Ungar and Foster, 1998] Ungar, L. and Foster, D. P. (1998). A Formal Statistical Approach to Collaborative Filtering. In In CONALD'98 CMU. 49

[Černý, 1985] Černý, V. (1985). Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. Journal of Optimization Theory and Applications *45*, 41–51. 93

[Wang et al., 2002] Wang, H., Wang, W., Yang, J. and Yu, P. S. (2002). Clustering by pattern similarity in large data sets. In SIGMOD '02: Proceedings of the 2002 ACM SIGMOD international conference on Management of data pp. 394–405, ACM, New York, NY, USA. 49

[Wolpert and Macready, 1997] Wolpert, D. H. and Macready, W. G. (1997). No free lunch theorems for optimization. IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION *1*, 67–82. 44

[Yang et al., 2003] Yang, J., Wang, H., Wang, W. and Yu, P. (2003). Enhanced Biclustering on Expression Data. In BIBE '03: Proceedings of the 3rd IEEE Symposium on BioInformatics and BioEngineering p. 321, IEEE Computer Society, Washington, DC, USA. 50, 61, 69, 109

[Yang et al., 2002] Yang, J., Wang, W., Wang, H. and Yu, P. (2002). delta-Clusters: Capturing Subspace Correlation in a Large Data Set. In Proc. of 18th IEEE Intern. Conf. on Data Engineering IEEE. 49

[Yip, 2003] Yip, K. (2003). DB Seminar Series: Biclustering Methods for Microarray Data Analysis. `www.cs.wayne.edu/~shiyong/csc7710/assignments/bicluster.ppt`. 46, 47

[Zitzler, 1999] Zitzler, E. (1999). Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications. Master's thesis Swiss Federal Institute of Technology Zurich. 40, 41

[Zitzler and Künzli, 2004] Zitzler, E. and Künzli, S. (2004). Indicator-based selection in multiobjective search. In in Proc. 8th International Conference on Parallel Problem Solving from Nature (PPSN VIII pp. 832–842, Springer. 42

[Zitzler et al., 2001] Zitzler, E., Laumanns, M. and Thiele, L. (2001). SPEA2: Improving the Strength Pareto Evolutionary Algorithm. Technical report Computer Engineering and Networks Laboratory - Swiss Federal Institute of Technology (ETH) Zurich. 42

Part VI

Appendices

Appendix A

Details of the biclusters presented for the Yeast dataset

The rows and columns that compose the biclusters exposed in subsection 4.2.2 are detailed in this appendix. The numbering of the rows and columns starts at 1.

Bicluster	**Genes**	**Conditions**
19	639, 678, 771, 1387, 2806	4, 6, 9, 10, 11, 12, 13, 14, 15, 16, 17
250	453, 1517, 1532, 1560, 1579, 1608, 1702, 1904, 1908, 1952, 2054, 2177	1, 4, 6, 7, 9, 10, 11, 13, 14, 15, 16, 17
251	407, 483, 564, 574, 629, 639, 741, 803, 805, 1107, 1204, 1226, 1387, 1408, 1435, 1600, 1673, 1750, 2026, 2573, 2649, 2662, 2717, 2724, 2806, 2830	1, 4, 6, 8, 9, 10, 11, 13, 15, 16, 17
385	657, 919, 949, 958, 972, 980, 1134, 2289, 2325, 2389, 2495, 2533, 2566, 2587, 2608	1, 6, 8, 9, 10, 11, 13, 14, 15, 16, 17
394	483, 564, 2662, 2724, 2814	1, 4, 5, 6, 8, 9, 10, 11, 13, 15, 16, 17
419	678, 975, 1848, 1862, 2097	1, 2, 4, 6, 8, 9, 10, 11, 13, 14, 15, 16, 17
439	611, 1147, 2325, 2365, 2587	1, 2, 4, 6, 7, 8, 9, 10, 11, 13, 14, 15, 16
453	123, 196, 295, 1649, 1750	1, 2, 4, 6, 9, 10, 11, 13, 14, 15, 16, 17
462	1560, 1608, 1625, 2109, 2455	1, 2, 6, 8, 9, 10, 11, 13, 14, 15, 16, 17
485	1137, 1538, 2042, 2144, 2427	1, 4, 6, 9, 10, 11, 13, 15, 16
497	678, 1643, 1690, 1788, 2023, 2097	1, 4, 5, 6, 9, 10, 11, 13, 15, 16, 17
526	314, 483, 1226, 1408, 2662	1, 3, 6, 8, 9, 10, 11, 13, 14, 15, 16, 17
540	92, 975, 1901, 1952, 2175	1, 3, 4, 6, 7, 9, 10, 11, 13, 14, 15, 16, 17
547	230, 472, 483, 564, 2814	1, 2, 5, 6, 7, 9, 10, 11, 13, 14, 16, 17

553	522, 574, 600, 657, 696, 735, 766, 1193, 1204, 1860, 2325, 2365, 2389, 2533, 2587, 2608	1, 3, 4, 6, 7, 9, 10, 11, 13, 14, 15, 16, 17

Appendix B

Details of the biclusters presented for the Human dataset

The rows and columns that compose the biclusters exposed in subsection 4.3.2 are detailed in this appendix. The numbering of the rows and columns starts at 1.

Bicluster	**Genes**	**Conditions**
36	2583, 2996, 3044, 3046, 3133	3, 4, 5, 7, 9, 10, 11, 14, 15, 18, 24, 33, 34, 36, 37, 38, 41, 42, 43, 46, 58, 59, 61, 62, 64, 65, 66, 68, 69, 71, 74, 75, 76, 77, 78, 79, 80, 86, 87, 88, 89, 90, 91, 93, 95
51	2665, 2722, 2930, 2973, 2990	2, 4, 12, 14, 17, 18, 20, 21, 23, 24, 25, 26, 28, 31, 32, 34, 35, 37, 38, 39, 40, 41, 42, 43, 45, 49, 50, 51, 53, 54, 58, 65, 69, 74, 76, 81, 83, 85, 88, 96
53	136, 1406, 1966, 2889, 3321	5, 6, 9, 10, 13, 15, 19, 26, 32, 35, 36, 41, 42, 43, 45, 46, 49, 52, 60, 62, 66, 68, 71, 72, 77, 83, 85, 96
121	1175, 2698, 3000, 3242, 3637	3, 6, 8, 9, 14, 16, 18, 19, 20, 22, 23, 24, 25, 26, 36, 37, 38, 40, 43, 48, 49, 53, 57, 59, 62, 64, 65, 66, 67, 69, 71, 72, 73, 76, 77, 78, 79, 81, 85, 87, 88, 89, 90, 91, 93
130	212, 1427, 1494, 1569, 1609, 1656, 1659, 1669, 2064, 2177, 2339, 2451, 2485, 2546, 2569, 2583, 2592, 2610, 2632, 2640, 2645, 2654, 2710, 2732, 2771, 2857, 2890, 2938, 2951, 2971, 2975, 3063, 3066, 3140, 3193, 3312, 3468, 3632	6, 7, 10, 11, 14, 16, 18, 19, 24, 37, 42, 49, 66, 68, 69, 76, 77, 78
144	2572, 2573, 2723, 2994, 3167	2, 4, 6, 9, 10, 11, 14, 17, 18, 19, 20, 21, 22, 31, 35, 38, 39, 42, 43, 45, 49, 50, 52, 54, 56, 58, 61, 65, 67, 69, 72, 74, 75, 77, 80, 84, 85, 89, 92

196	1520, 2577, 2581, 2583, 2946, 2982, 2996, 3000, 3001, 3039, 3041, 3044, 3060, 3061, 3067, 3068, 3133, 3151, 3164, 3168, 3210, 3217, 3250, 3309, 3628, 3637, 3667, 3673, 3956	4, 5, 7, 8, 9, 10, 13, 14, 15, 18, 22, 25, 26, 28, 32, 33, 36, 37, 38, 42, 43, 44, 45, 47, 51, 67, 68, 71, 72, 75, 76, 77, 78, 79, 81, 83, 84, 85, 87, 88, 89, 90, 91, 96
217	159, 2577, 2612, 2990, 3018	1, 4, 5, 6, 7, 8, 9, 10, 14, 18, 19, 22, 23, 24, 26, 31, 33, 34, 37, 40, 41, 42, 44, 45, 46, 49, 50, 53, 58, 59, 62, 64, 65, 66, 68, 69, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 83, 84, 85, 86, 89
237	2665, 2722, 2723, 2872, 2960	1, 2, 3, 5, 7, 10, 19, 20, 21, 24, 25, 26, 32, 38, 40, 42, 43, 45, 46, 49, 58, 60, 61, 62, 65, 68, 70, 71, 75, 76, 77, 79, 80, 81, 84, 85, 89, 90, 91, 93, 96
252	2583, 2609, 2698, 2846, 2872, 2996, 3000, 3046, 3133, 3143, 3637, 3954, 3984	3, 5, 7, 9, 10, 14, 15, 18, 19, 21, 22, 24, 27, 34, 35, 36, 37, 38, 39, 42, 43, 46, 49, 53, 59, 74, 75, 76, 77, 78, 79, 80, 87, 89, 96
292	2553, 3041, 3187, 3223, 3293	4, 5, 6, 7, 8, 9, 14, 15, 16, 22, 24, 25, 28, 33, 34, 38, 43, 45, 48, 49, 54, 60, 67, 68, 74, 75, 84, 85, 87, 90, 92
314	135, 2982, 2996, 3041, 3133	4, 6, 7, 8, 9, 11, 15, 16, 17, 18, 19, 25, 26, 28, 34, 36, 37, 38, 40, 43, 51, 52, 53, 57, 59, 65, 66, 67, 68, 74, 76, 77, 78, 80, 82, 84, 85, 87, 89, 90, 91, 92, 96
318	206, 2996, 3651, 3667, 3953	5, 7, 8, 10, 11, 16, 17, 18, 22, 26, 28, 33, 36, 37, 38, 40, 41, 43, 46, 49, 52, 68, 76, 79, 81, 85, 89, 96
331	2612, 2621, 2665, 2722, 2723	7, 13, 14, 17, 21, 22, 24, 26, 29, 30, 31, 32, 34, 37, 38, 40, 42, 43, 45, 50, 51, 53, 55, 65, 81, 85, 88, 89, 90, 93, 95
363	456, 2377, 2565, 2645, 3193	3, 5, 6, 7, 8, 9, 10, 11, 13, 14, 38, 39, 42, 43, 45, 48, 49, 50, 55, 58, 64, 65, 66, 68, 71, 74, 78, 95
412	212, 2553, 2573, 3168, 3637	3, 4, 5, 6, 10, 11, 15, 17, 18, 19, 21, 22, 24, 28, 34, 35, 38, 42, 43, 45, 50, 54, 62, 65, 66, 73, 76, 77, 90, 92, 96
436	2473, 2640, 2641, 2982, 3133	4, 5, 6, 7, 9, 10, 11, 14, 15, 20, 24, 34, 36, 39, 46, 49, 50, 51, 53, 54, 57, 58, 62, 66, 71, 72, 74, 75, 76, 77, 79, 84, 85, 88, 89, 90, 91, 93, 95

448	2577, 2722, 2990, 3018, 3157	4, 5, 7, 8, 9, 13, 14, 18, 19, 22, 23, 25, 26, 31, 32, 33, 34, 35, 37, 40, 41, 42, 46, 48, 50, 64, 67, 68, 69, 72, 73, 83, 84
465	2573, 2648, 3041, 3193, 3250	1, 2, 4, 5, 6, 7, 8, 9, 13, 14, 15, 17, 18, 19, 22, 30, 31, 36, 37, 38, 39, 40, 42, 43, 44, 49, 50, 53, 54, 58, 60, 61, 62, 72, 73, 76, 77, 78, 79, 81, 87, 88, 90, 92, 95
480	2866, 2882, 2883, 2884, 3045	7, 8, 9, 10, 11, 14, 15, 16, 18, 19, 20, 21, 22, 24, 26, 29, 31, 34, 35, 36, 37, 38, 42, 43, 45, 46, 48, 54, 57, 58, 59, 60, 61, 65, 69, 71, 72, 75, 76, 81, 83, 88, 89, 95, 96
492	1175, 2996, 3040, 3041, 3046	4, 5, 6, 12, 15, 17, 18, 19, 22, 25, 30, 33, 35, 37, 38, 42, 43, 46, 53, 60, 61, 64, 65, 71, 72, 74, 76, 77, 78, 81, 90, 93, 96
519	134, 666, 1054, 2358, 2359	7, 8, 9, 10, 11, 14, 20, 24, 25, 31, 32, 35, 36, 37, 38, 42, 44, 45, 49, 50, 54, 60, 63, 65, 66, 69, 71, 73, 79, 80, 89
553	1175, 2448, 2971, 3231, 3250	3, 9, 10, 18, 31, 32, 33, 35, 36, 37, 42, 43, 54, 58, 62, 65, 68, 71, 73, 74, 75, 76, 77, 78, 80, 81, 83, 89, 95, 96
576	2698, 2982, 3046, 3133, 3167	3, 5, 7, 9, 10, 13, 19, 25, 32, 36, 42, 45, 46, 48, 49, 61, 63, 66, 68, 69, 71, 74, 75, 76, 77, 80, 81, 91, 92